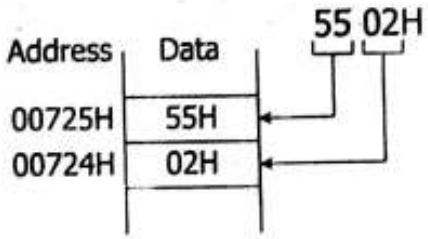
## Intel 8086 ( Memory Address Space & Data Organization)

- The memory space is organized as individual bytes of data stored at consecutive addresses over the address range $00000_{16}$ to $FFFFF_{16}$ (or 00000H to FFFFFH)

- Two consecutive bytes can be accessed as one word (16 bits) of data

- The **lower-addressed byte** is the **least significant byte** of the word, and the **higher-addressed byte** is its **most significant byte**.

- For e.g. Storing 1 word 5502H in the memory:




- Words of data can be stored at either even or odd-address boundary.

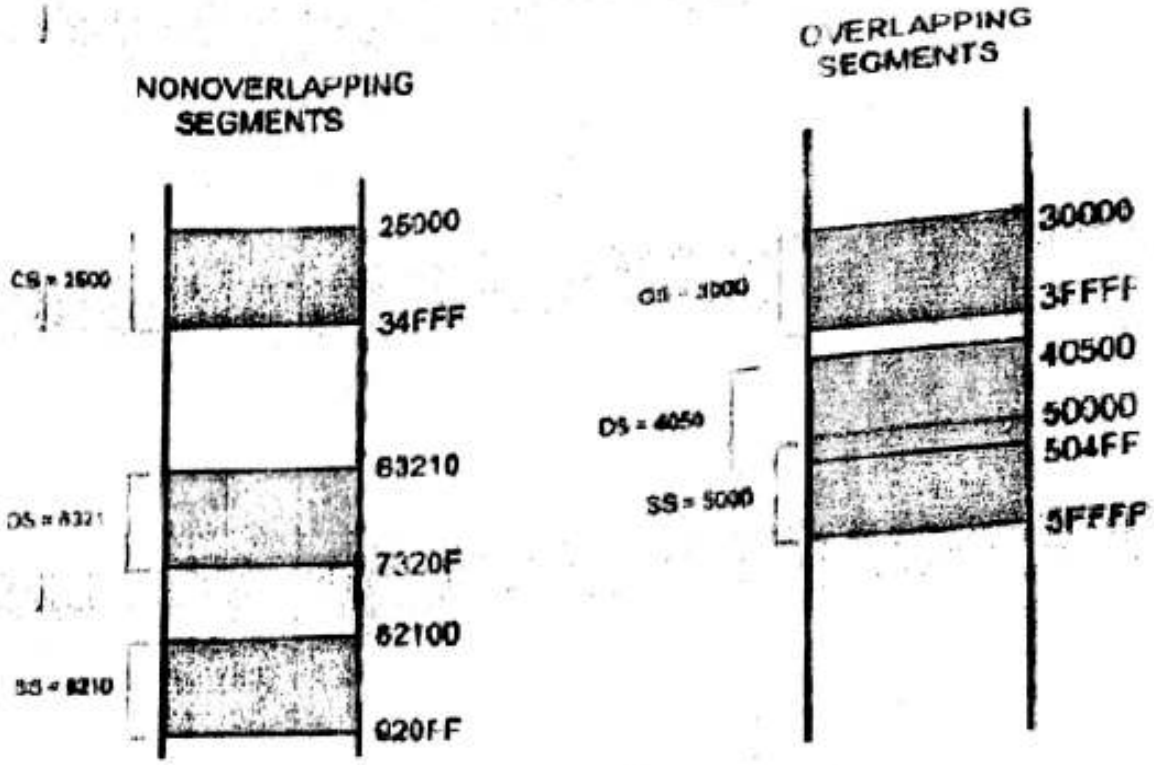
- A word stored at an even-address boundary (00000H, 00002H, 00004H etc) is said to be an **aligned word**

- A word stored at an odd address boundary (00001H, 00003H, 00005H etc) is called **misaligned word**

- In calculating the physical address, it is possible that two segments can overlap, which is desirable in some circumstances.



Scanned by CamScanner




- A single **physical address** can belong to many different **logical addresses.** E. g.

| Logical address (hex) | Physical address (hex) |
|---|---|
| 1000:5020 | 15020 |
| 1500:0020 | 15020 |
| 1502:0000 | 15020 |
| 1400:1020 | 15020 |
| 1302:2000 | 15020 |



Scanned by CamScanner

Example

If CS = 24F6H and IP = 634AH, show:

(a) The logical address

(b) The offset address

And calculate:

(c) The physical address

(d) The lower range

(e) The upper range of the code segment

Solution:

(a) 24F6:634A

(b) 634A

(c) 2B2AA (24F60 + 634A)

(d) 24F60 (24F60 + 0000)

(e) 34F5F (24F60 + FFFF)

Example

Assume that the DS register is 578C. To access a given byte of data at physical memory location 67F66, does the data segment cover the range where the data is located? If not, what changes need to be made?

Solution:

No, since the range is 578C0 to 678BF, location 67F66 is not included in this range. To access that byte, DS must be changed so that its range will include that byte.



Scanned by CamScanner

Example

Assume memory locations with the following contents: DS:6826 = 48 and DS:6827 = 22. Show the contents of register BX in the instruction MOV BX, [6826]

Solution:

DS:6826 = 48

DS:6827 = 22

| BH | BL |
|---|---|
| 22 | 48 |

Example

Show how the flag register is affected by

```
MOV   AL, 9CH     ;AL = 9CH
MOV   DH, 64H     ;DH = 64H
ADD   AL, DH      ;now AL = 0
```

Solution:

```
      9C          1001 1100
+     64          0110 0100
      00          0000 0000
```

CF = 1 since there is a carry beyond d7

PF = 1 since there is an even number of 1s in the result

AF = 1 since there is a carry from d3 to d4

ZF = 1 since the result is zero

SF = 0 since d7 of the result is zero



Scanned by CamScanner

Example
Show how the flag register is affected by the addition of 38H and 2FH.

Solution:

```
        MOV   BH, 38H        ;BH = 38H
        ADD   BH, 2FH        ;add 2F to BH, now BH = 67H
```

```
          38        0011 1000
    +     2F        0010 1111
          67        0110 0111
```

CF = 0 since there is no carry beyond d7
PF = 0 since there is an odd number of 1s in the result
AF = 1 since there is a carry from d3 to d4
ZF = 0 since the result is not zero
SF = 0 since d7 of the result is zero

Example
Show how the flag register is affected by

```
        MOV   AX, 34F5H      ;AX = 34F5H
        ADD   AX, 95EBH      ;now AX = CAE0H
```

Solution:

```
          34F5      0011 0100 1111 0101
    +     95EB      1001 0101 1110 1011
          CAE0      1100 1010 1110 0000
```

CF = 0 since there is no carry beyond d15
PF = 0 since there is an odd number of 1s in the lower byte
AF = 1 since there is a carry from d3 to d4
ZF = 0 since the result is not zero
SF = 1 since d15 of the result is one



Scanned by CamScanner

Example

Show how the flag register is affected by

```
MOV   AX, AAAAH      ;BX = AAAAH
ADD   AX, 5556H      ;now BX = 0000H
```

Solution:

```
   AAAA        1010 1010 1010 1010
+  5556        0101 0101 0101 0110
   0000        0000 0000 0000 0000
```

CF = 1 since there is no carry beyond d15

PF = 1 since there is an even number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 1 since the result is zero

SF = 0 since d15 of the result is zero

**Exercise**

Given SP = A238H, SS = 0107H , AX= ABCDH and an instruction PUSH AX. Determine the following:

a. TOS before executing the instruction.

b. TOS after executing the instruction.

c. Content of SP register.

d. The logical address into which AB is pushed.

e. The physical address into which CD is pushed.



Scanned by CamScanner

# HW#2

1- What you mean by pipelining in 8086 processor ?

2- How the 20 bit effective address is calculated in 8086 processor?

3- Compare 8085 and 8086 processor?

4- Give the flag format of 8086?

5- What is the function of direction flag?

6- What are the function of segment register?

7- Explain memory organization in 8086?

8- Draw and explain the architecture of 8086?

9- What is physical address?

10- Define OFFDET address?

11- What are the versions of 8086?

12- List the functions of Bus Interface Unit in 8086?

13- Write about the auxiliary carry flag used in 8086 ?

14- Is overlapping of segments possible in 8086?

15- What is Packed BCD Format?

16- Why in memory segmentation of 8086 it use 64Kb instead of 1Mb?

17- Mention the jobs performed by EU?

18- Explain the operations of instructions queue residing in BIU.

19- Mention the total number of registers of 8086 and show the manner in which they are grouped?

20- Describe, in detail, the general purpose of data registers?



Scanned by CamScanner

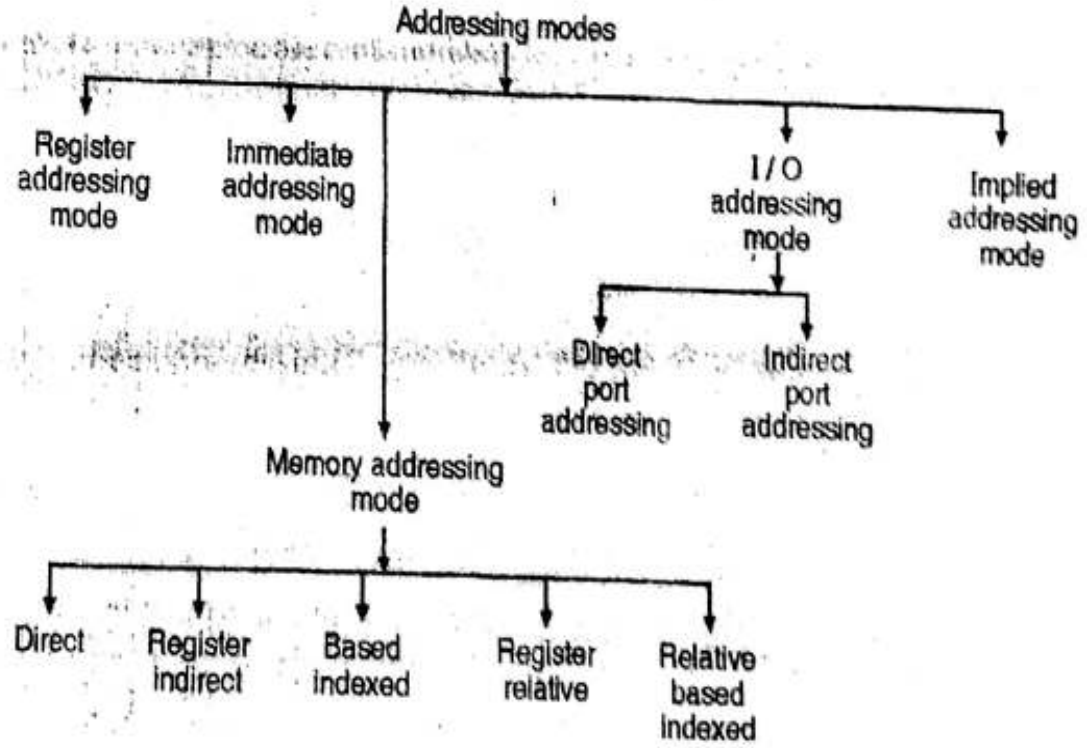
# Addressing modes of 8086

- When 8086 executes an instruction, it performs the specified function on data. these data are called its **operands** and may be ***part of the instruction***, reside in one of the ***internal registers*** of the microprocessor, stored at an address in ***memory*** or held at an ***I/O port***.

- The set of mechanisms by which an instruction can specify how to obtain its operands is known as **Addressing modes**. The CPU can access the operands (data) in a number of different modes.

The addressing modes available in Intel 8086 are:

1. Register Addressing
2. Immediate Addressing
3. Implied Addressing
4. Direct Addressing
5. Register Indirect Addressing
6. Based Relative Addressing
7. Indexed Relative Addressing
8. Based Indexed Relative Addressing



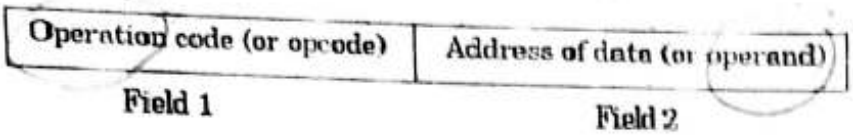



Scanned by CamScanner

# Opcode and Operand in Microprocessor 8086/8088

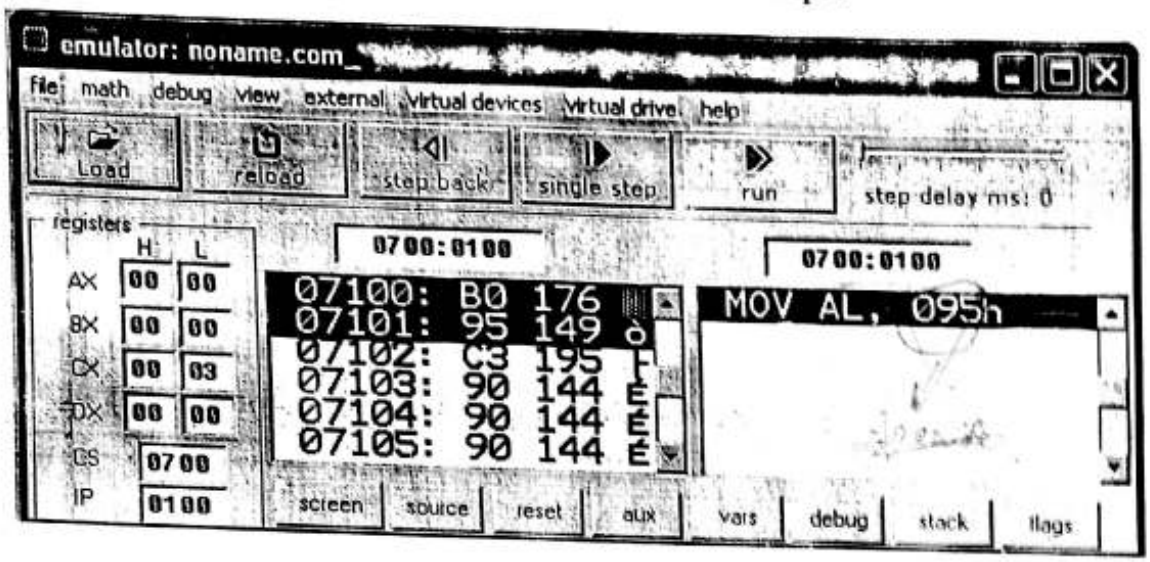
**What an instruction consists of?**

An instruction consists of an operation code (called '**opcode**') and the address of the data (called 'operand'), on which the opcode operates.




An opcede executed by the CPU. In machine language it is a binary or hexadecimal value such as **B0** loaded into the instruction register. In assembly language mnemonic form an opcode is a command such as MOV or ADD or JMP.

**Example**: MOV AL, 95H ; MOV is the opcode. ; AL (register) is an operand. Operands are manipulated by the opcode. In this example




MOV AX, 1000H ; MOV is the opcode. ;

AX (register) is an operand.

Operands are manipulated by the opcode. In this example, the operands are the register AL and the value 95H.

| Op code | Operand |
|---|---|
| 1- byte | 1 or 2 byte |

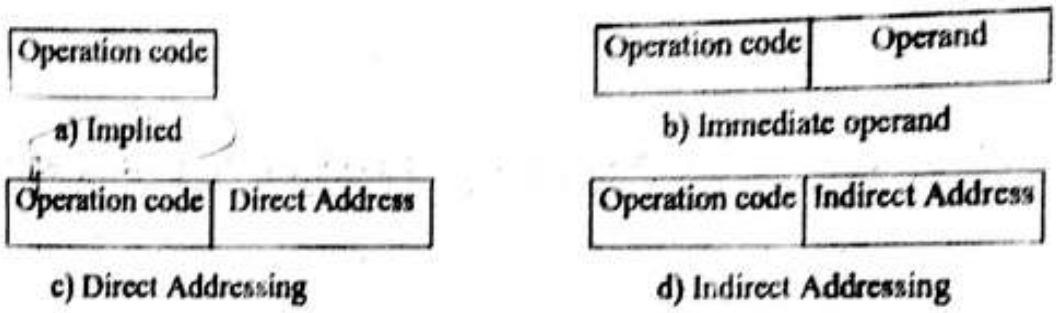


Scanned by CamScanner

**Instructions consist of:**

- **operation** (opcode) e.g. MOV
- **operands** (number depends on operation)
- operands specified using addressing modes.
- addressing mode may include **addressing information** e.g. registers, constant values
- Encoding of instruction must include opcode, operands & addressing information.

**What are different instruction formats?**




**What is opcode fetch cycle?**

The opcode fetch cycle is a machine cycle executed to fetch the opcode of an instruction stored in memory. Each instruction starts with opcode fetch machine cycle.

**What is meant by addressing mode?**

An instruction consists of an opcode and an operand. The operand may reside in the accumulator, or in a general purpose register or in a memory location. The manner in which an operand is specified (or referred to) in an instruction is called addressing mode.

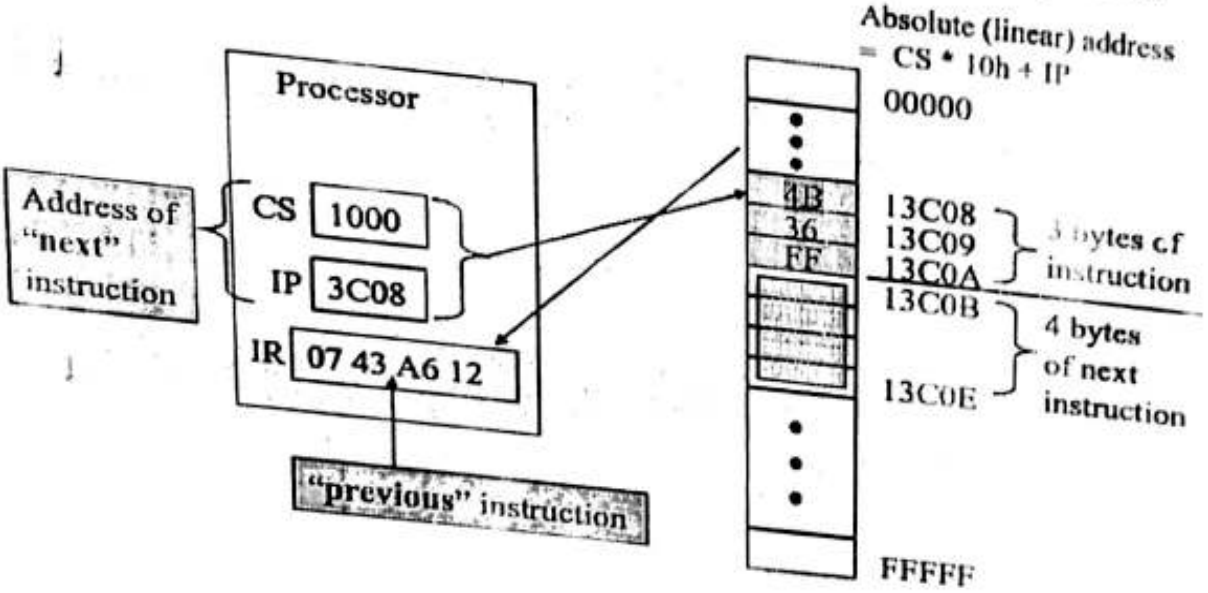


Scanned by CamScanner

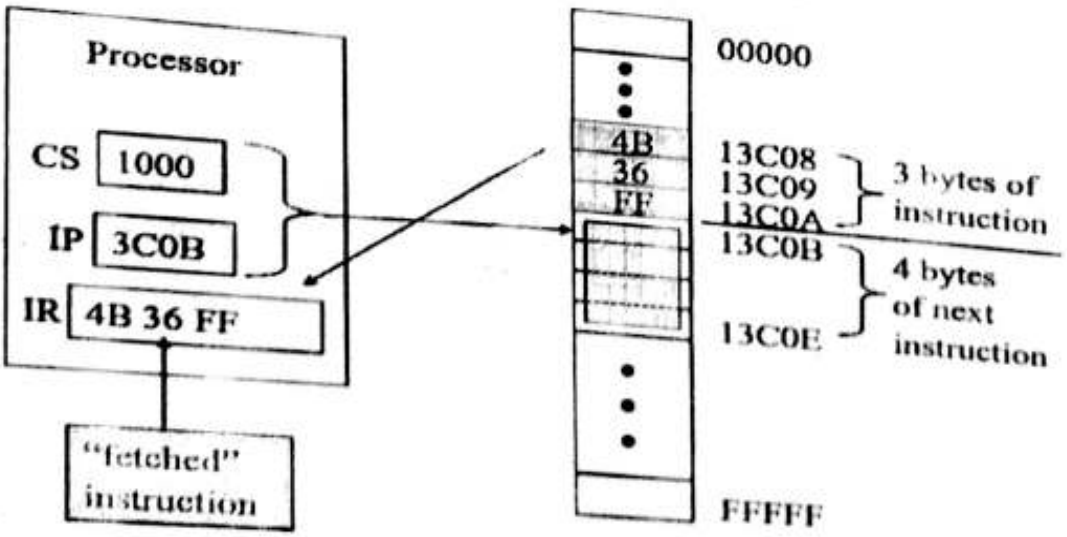
**The first byte (opcode) of instruction tells the number of bytes to be fetched**

## Let's refine the Instruction Execution Cycle ...

Before fetch:




After fetch:



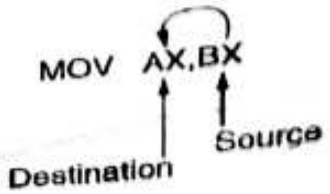



Scanned by CamScanner

## Data Addressing Mode

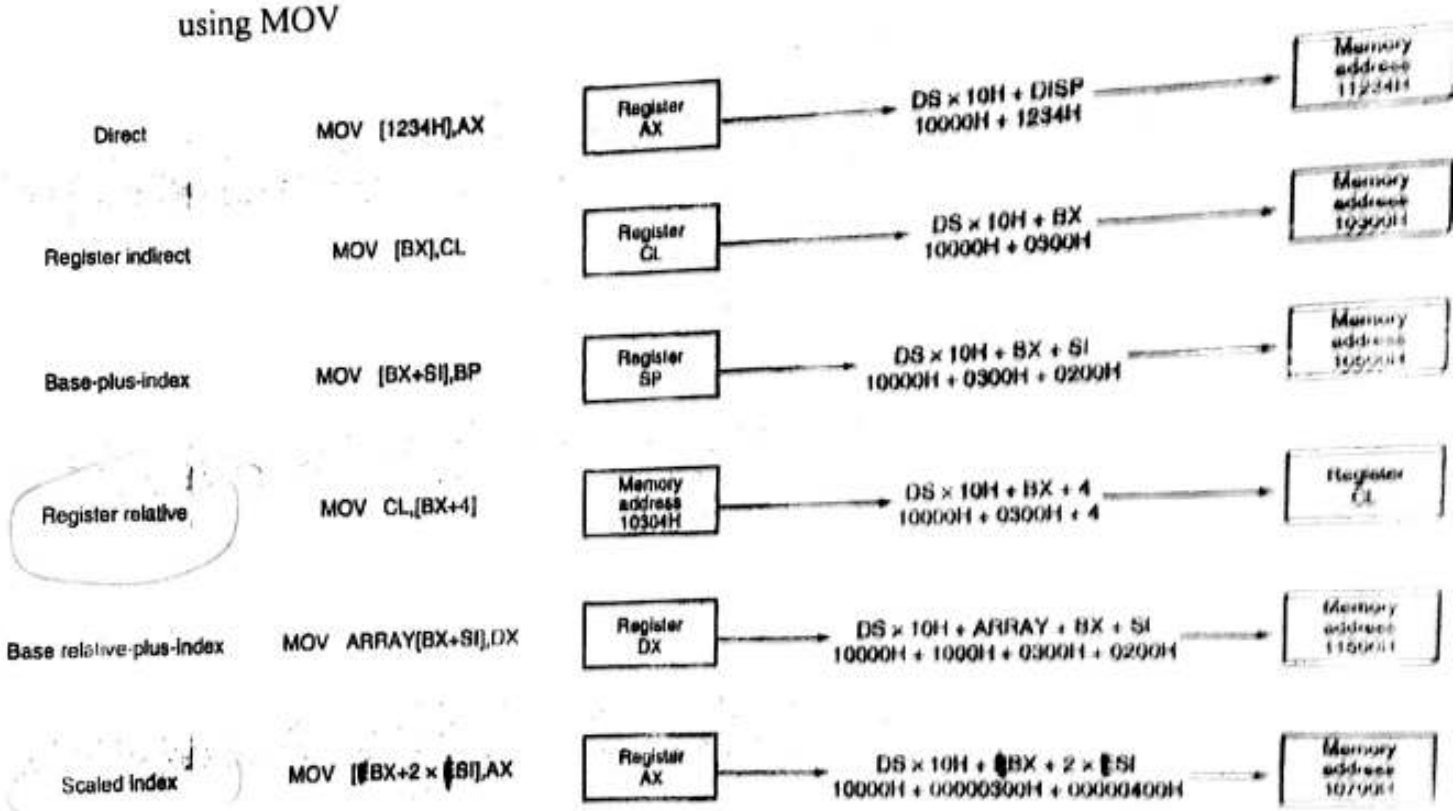
- MOV instruction is a common and flexible instruction. Provide a basis for explanation of data addressing modes.

- The following figure illustrates the MOV instruction and defines the direction of data flow.




- Source is to the right and destination the left, next to the OPCODE MOV.

- The figure bellow shows all possible variation of the data-addressing mode using MOV




Notes: BX = 0300H, SI = 0200H, ARRAY = 1000H, and DS = 1000H



Scanned by CamScanner

**1. Register addressing mode:**

- involves the use of registers
- memory is not accessed, so faster
- source and destination registers must match in size.
- Data is in register and Instruction Specifies the particular register

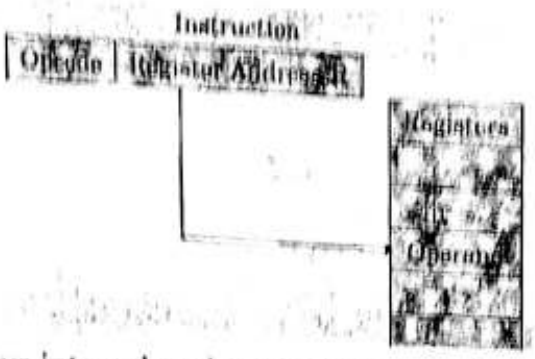



Fig. below shows internal registers, anyone can be used as a source or destination operand, however only the data registers can be accessed as either a byte or word.

| *Assembly Language* | *Size* | *Operation* |
|---|---|---|
| MOV AL,BL | 8-bits | Copies BL into AL |
| MOV CH,CL | 8-bits | Copies CL into CH |
| MOV AX,CX | 16-bits | Copies CX into AX |
| MOV SP,BP | 16-bits | Copies BP into SP |
| MOV DS,AX | 16-bits | Copies AX into DS |
| MOV SI,DI | 16-bits | Copies DI into SI |

**Note:** Instruction which affect only registers execute faster than instruction s involving memory access, as no time is spent fetching operands.

Example: INC BX , DEC DX , SUB DX,CX

***Example***

```
0000 8B C3          MOV AX,BX       ;copy contents of BX into AX
0002 8A CE          MOV CL,DH       ;copy the contents of DH into CL
0004 8A CD          MOV CL,CH       ;copy the contents of CH into CL
0006 8C C8          MOV AX,CS       ;
0008 8E D8          MOV DS,AX       ;   copy CS into DS
```



Scanned by CamScanner

Note
A constant such as
directive such as

| | | Before | After |
|---|---|---|---|
| Ex1: MOV CX, SI | CX | 1234H | 5678H |
| | SI | 5678H | 5678H |

| | | Before | After |
|---|---|---|---|
| Ex2: MOV DL, AH | DI | 89H | BCH |
| | AH | BCH | BCH |

## *2. Immediate addressing mode:*

- source operand is a constant.
- Immediate addressing mode can be used to load information into any of the registers except the segment registers and flag registers.
- The operand comes immediately after the opcode. For this reason, this addressing mode executes quickly.
- In this mode, 8 or 16 bit data can be specified as part of the instruction.

| OP Code | Immediate Operand |
|---|---|

**Examples of immediate addressing using the MOV instruction**

```
MOV  BL,44      8-bits Copies a 44 decimal (2CH) into BL
MOV  AX,44H     16-bits Copies a 0044H into AX
MOV  SI,0       16-bits Copies a 0000H into SI
MOV  CH,100     8-bits Copies a 100 decimal (64H) into CH
MOV  AL,'A'     8-bits Copies an ASCII A into AL
```

```
0100  B8 0000          MOV     AX,0          ;place 0000H into AX
0103  BB 0000          MOV     BX,0000H      ;place 0000H into BX
0106  B9 0000          MOV     CX,0          ;place 0000H into CX

0109  8B F0            MOV     SI,AX         ;copy AX into SI
010B  8B F8            MOV     DI,AX         ;copy AX into DI
010D  8B E8            MOV     BP,AX         ;copy AX into BP
```

| | | Before | After |
|---|---|---|---|
| Ex1: MOV DX, 1234H | DX | ABCDH | 1234H |

| | | Before | After |
|---|---|---|---|
| Ex2: MOV CH, 23H | CH | 4DH | 23H |



Scanned by CamScanner

*Note*

A constant such as "VALUE" can be defined by the *assembler* EQU *directive* such as

```
VALUE EQU 38H
MOV BH , VALUE
```

***3. Implied addressing mode:***

Instruction using this mode have no operands.

**Example :**

CLC — which clears carry flag to zero.

STD — which set Direction flag to one.

***4. Direct addressing mode:***

- Address of the data in memory comes immediately after the instruction operand is a constant.
- The default segment is always DS. The 20 bit physical address of the operand in memory is normally obtained as PA = DS : EA
- The address is the offset address. The offset address is put in a rectangular bracket.

**Example of Direct addressing mode using Mov Instruction:**

**MOV AL,[2C00h]**

8-bits Copies the byte contents of data segment memory location 2C00h into AL. The physical address is calculated by combining the contents of offset location 2C00 with DS

**MOV AX,[3400h]**

16-bits Copies the word contents of data segment memory location 3400h into AX

**MOV [100h],BL**

8-bits Copies BL into data segment memory location 100h

**MOV [72C2h],CX**

16-bits Copies CX into data segment memory location 72C2h

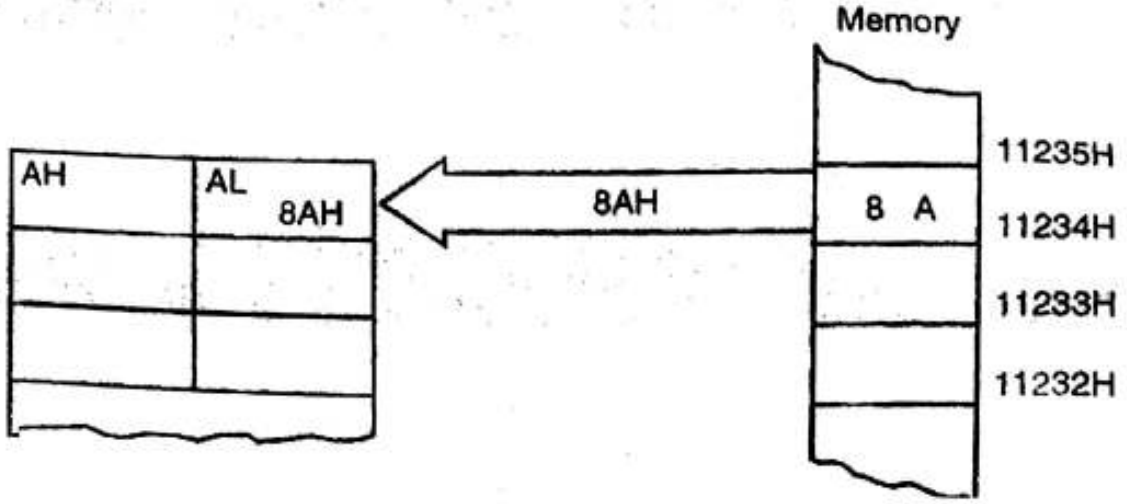
**MOV ES:[2000 H],AL**

8-bits Copies AL into extra Data segment memory location 2000H

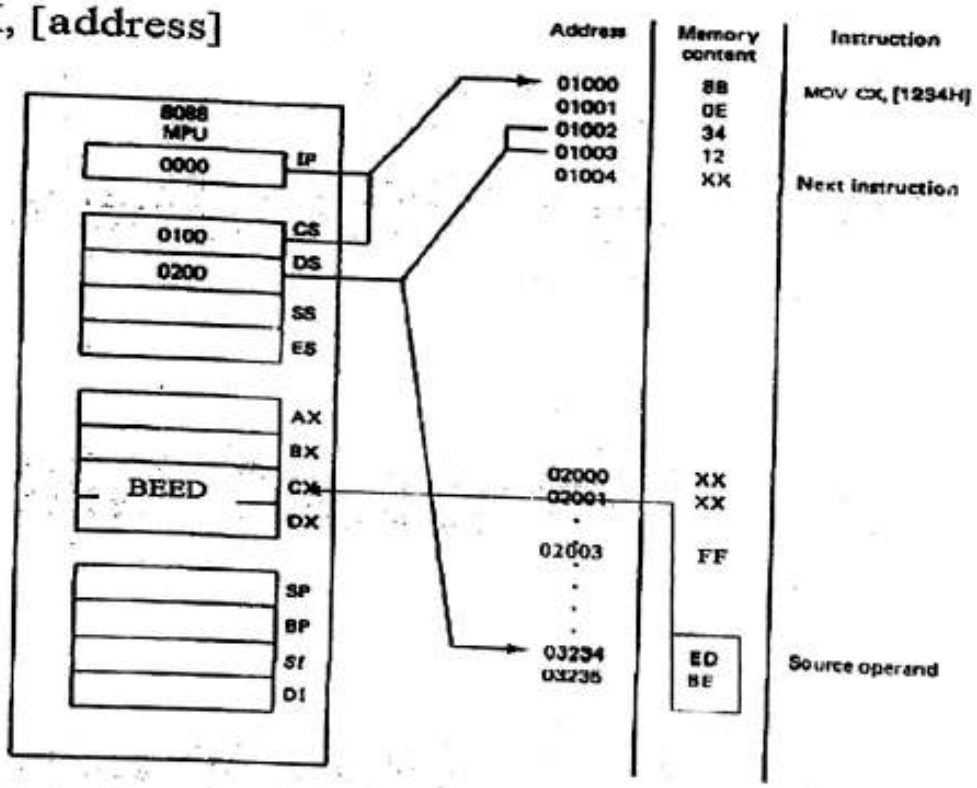


Scanned by CamScanner

MOV AL,[1234H]  DS = 1000H



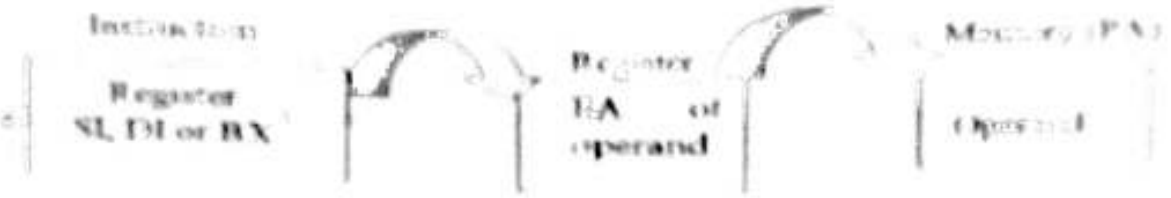

MOV CX, [address]

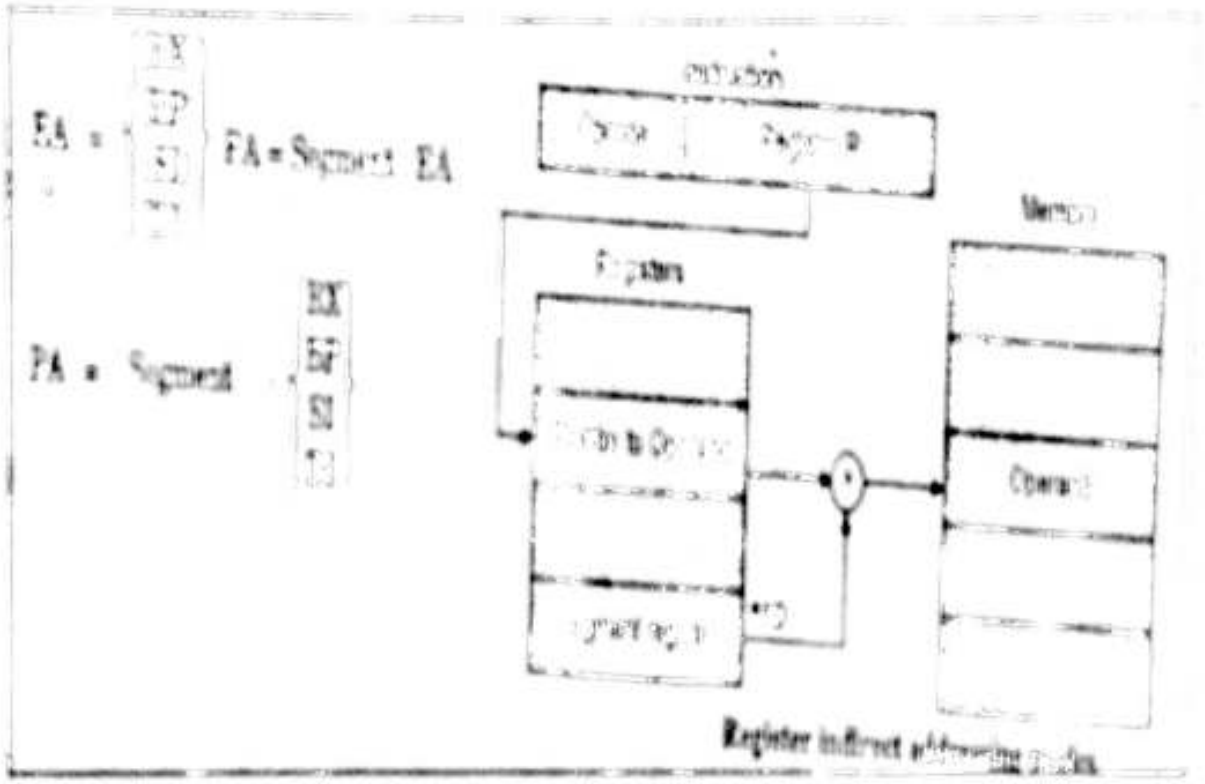




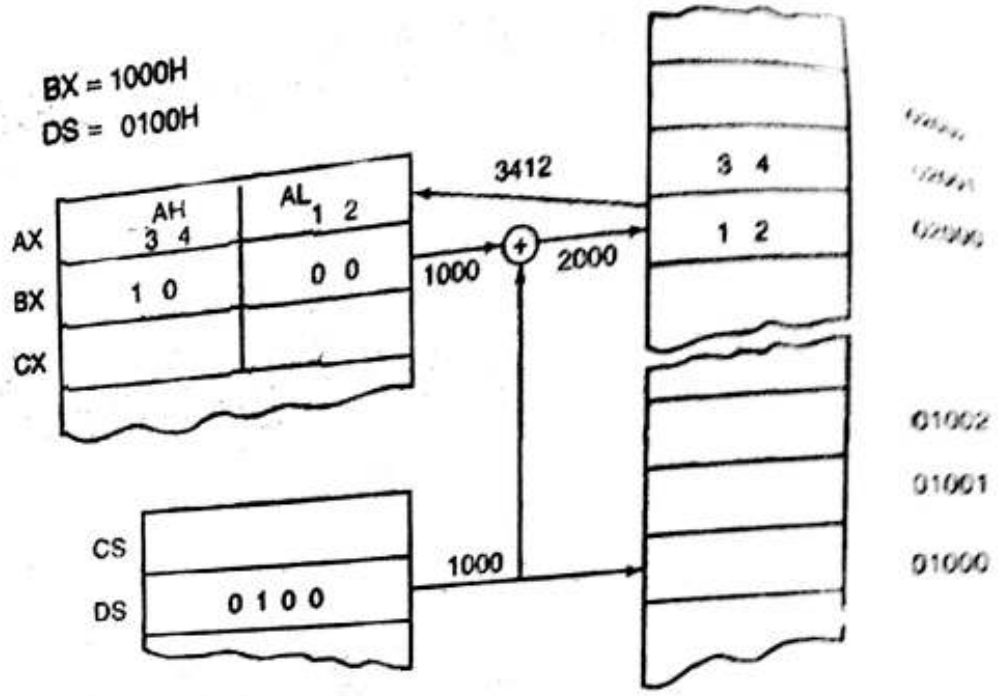

Scanned by CamScanner

5. *Register indirect addressing mode.*



MOV AX,[BX]




Ex1: MOV CL, [SI]

| | Before | After |
|---|---|---|
| CL | ~~20H~~ | 78H |
| SI | 3456H | |
| DS:3456H | 78H | |

Ex2: MOV DX, [BX]

| | Before | After |
|---|---|---|
| DX | ~~F232H~~ | 3567H |
| BX | A2B2H | |
| DS:A2B2H | 67H | LS byte |
| DS:A2B3H | 35H | MS byte |

Ex3: MOV AH, [DI]

| | Before | After |
|---|---|---|
| AH | ~~30H~~ | 86H |
| DI | 3400H | |
| DS:3400H | 86H | |



Scanned by CamScanner

### 6. Based relative addressing mode:

- BX and BP are known as the base registers. In this mode base registers as well as a displacement value are used to calculate the effective address.

- The default segments used for the calculation of Physical address (PA) are DS for BX, and SS for BP.

Ex: MOV CX,[BX]+10 ; move DS:BX+10 and DS:BX+11 into CX

; PA = DS (shifted left) +BX+10

- Note that, the content of the low address will go into CL and the high address contents will go into CH.

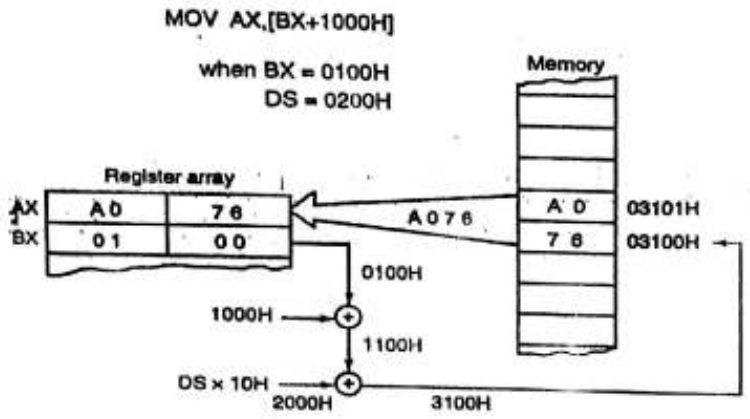
- There is alternative coding:

MOV CX,[BX+10],

MOV CX,10[BX]

- BX+10 is effective address

Ex: MOV AL,[BP]+5 ; PA = SS (shifted left) +BP+5






Scanned by CamScanner

**Based Addressing with displacement**

Ex1: MOV DH, 2345H[BX]

2345H is 16-bit displacement

4000 + 2345 = 6345H

| | Before | After |
|---|---|---|
| DH | ~~45H~~ | 67H |
| BX | 4000H | |
| DS:6345H | 67H | |

Ex2: MOV AX, 45H[BP]

45H is 8-bit displacement

3000 + 45 = 3045H
It is SS when BP is used

| | Before | After | |
|---|---|---|---|
| AX | ~~1000H~~ | CDABH | |
| BP | 3000H | | |
| SS:3045H | ABH | | LS byte |
| SS:3346H | CDH | | MS byte |

## *7. Indexed relative addressing mode:*

- Indexed relative addressing mode works the same as the based relative addressing mode.
- Except the registers DI and SI holds the offset address.

Ex:
```
MOV DX,[SI]+5      ;PA=DS(shifted left)+SI+5
MOV CL,[DI]+20     ;PA=DS(shifted left)+DI+20
```

**Indexed Addressing with displacement**

Ex1: MOV CL, 2345H[SI]

2345H is 16-bit displacement

6000 + 2345 = 8345H

| | Before | After |
|---|---|---|
| CL | ~~60H~~ | 85H |
| SI | 6000H | |
| DS:8345H | 85H | |

Ex2: MOV DX, 37H[DI]

37H is 8-bit displacement

5000H+ 37H = 5037H

| | Before | After | |
|---|---|---|---|
| DX | ~~7000H~~ | B2A2H | |
| DI | 5000H | | |
| DS:5037H | A2H | | LS byte |
| DS:5038H | B2H | | MS byte |

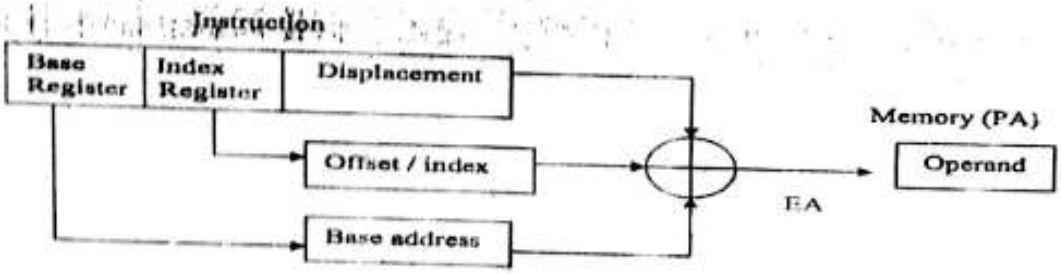


Scanned by CamScanner

## 8- Based Base relative Indexed addressing mode: (Relative Based - Indexed Mode)

- The combination of the based and indexed addressing modes.
- One base register and one index register are used.

- Default segment register for memory is DS. Only BX, BP and SI, DI registers can be used.




EA = {(BX) or (BP)} + {(SI) or (DI)} + {8-bit sign-extended displacement or 16-bit signed displacement}

PA = EA + (DS * 10H)

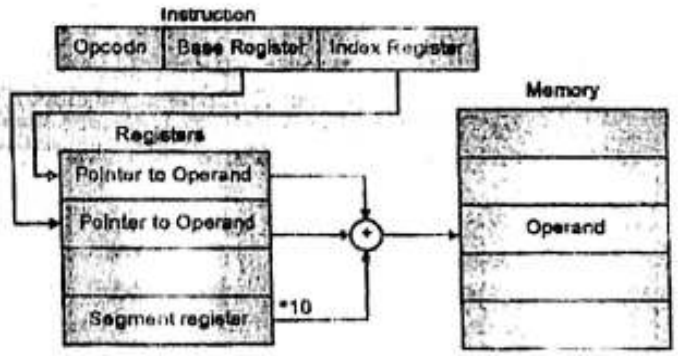
Ex: MOV 50h [BX] [SI] , 1234H  MOV -30H [BP] [SI], DX

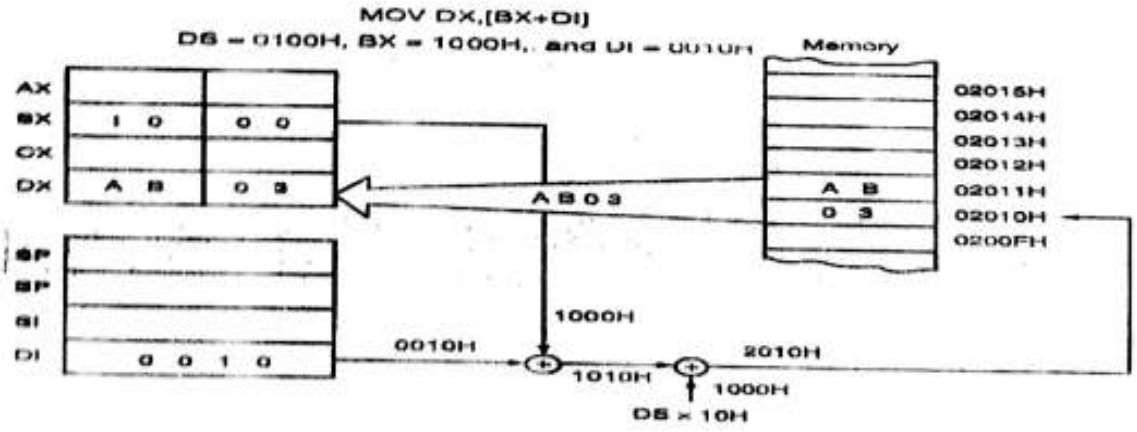
∴EA = (Base register) + (Index register)

$= \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}$

PA = Segment register : EA

$= \begin{Bmatrix}DS\\SS\end{Bmatrix} : \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}$



## Base-Plus-Index Addressing Mode: Examples

| *Assembly Language* | *Size* | *Operation* |
|---|---|---|
| MOV CX,[BX+DI] | 16-bits | Copies the word contents of the data segment memory location address by BX plus DI into CX |
| MOV CH,[BP+SI] | 8-bits | Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH |
| MOV [BX+SI],SP | 16-bits | Copies SP into the data segment memory location addresses by BX plus SI |
| MOV [BP+DI],AH | 8-bits | Copies AH into the stack segment memory location addressed by BP plus DI |

**Based Indexed Addressing with Displacement**

Ex1: MOV DL, 37H[BX+DI]

37H is 8-bit displacement

2000H + 0050H + 37H = 2300H

| | Before | After |
|---|---|---|
| DL | ~~40H~~ | 12H |
| BX | 2000H | |
| DI | 0050H | |
| DS:2087H | 12H | |

Ex2: MOV BX, 1234H[SI+BP]

4000H + 0020H +1234 = 5254H
It is SS when BP is used

| | Before | After | |
|---|---|---|---|
| BX | ~~3000H~~ | 3665H | |
| SI | 4000H | | |
| BP | 0020H | | |
| SS:5254H | 65H | | LS byte |
| SS:5255H | 36H | | MS byte |

```
Ex:   MOV CL,[BX][DI]+8     ;PA=DS(shifted left)+BX+DI+8
      MOV CH,[BX][SI]+20    ;PA=DS(shifted left)+BX+SI+20
      MOV AH,[BP][DI]+12    ;PA=SS(shifted left)+BP+DI+12
      MOV AL,[BP][SI]+29    ;PA=SS(shifted left)+BP+SI+29
```



Scanned by CamScanner

Alternative coding
MOV CL,[BX+DI+8]
MOV CL,[DI+BX+8]

Ex : MOV CL, [SI][BX]

| | Before | After |
|---|---|---|
| CL | ~~40H~~ | 67H |
| SI | 2000H | |
| BX | 0300H | |
| DS:2300H | 67H | |

2000H + 0300H = 2300H

Ex : MOV CX, [BP][DI]

| | Before | After |
|---|---|---|
| CX | ~~6000H~~ | 6385H |
| BP | 3000H | |
| DI | 0020H | |
| SS:3020H | 85H | LS byte |
| SS:3021H | 63H | MS byte |

It is SS when BP is used

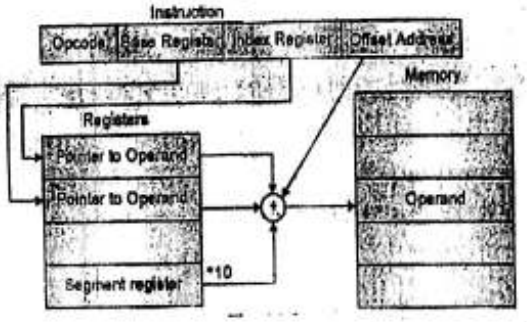
## MOV CX, [BX+SI+0400]

$$EA = \{\text{Base register}\} + \{\text{Index register}\} + \begin{Bmatrix} \text{8 bit displacement} \\ \text{(sign extended)} \\ \text{16 bit displacement} \end{Bmatrix}$$

$$= \begin{Bmatrix} (BX) \\ (BP) \end{Bmatrix} + \begin{Bmatrix} (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8/16 bit} \\ \text{displacement} \end{Bmatrix}$$

$$PA = \text{Segment register} : EA = \begin{Bmatrix} DS \\ SS \end{Bmatrix} : \begin{Bmatrix} (BX) \\ (BP) \end{Bmatrix} + \begin{Bmatrix} (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8/16 bit} \\ \text{displacement} \end{Bmatrix}$$

| Generation of EA | | Generation of PA | |
|---|---|---|---|
| 1000H | → [BX] | 23140H | → [DS] |
| + 1100H | → [SI] | + 2500H | → EA |
| + 0400H | → Displacement | 25640H | → PA |
| 2500H | → EA | | |




*The contents of location 25640H will be transferred to the CL register and the contents of location 25641H will be transferred to the CH register.*



Scanned by CamScanner

# Summary of the addressing modes

| Addressing Mode | Operand | Default Segment |
|---|---|---|
| Register | Reg | None |
| Immediate | Data | None |
| Direct | [offset] | DS |
| Register Indirect | [BX]<br>[SI]<br>[DI] | DS<br>DS<br>DS |
| Based Relative | [BX]+disp<br>[BP]+disp | DS<br>SS |
| Indexed Relative | [DI]+disp<br>[SI]+disp | DS<br>DS |
| Based Indexed Relative | [BX][SI or DI]+disp<br>[BP][SI or DI]+disp | DS<br>SS |

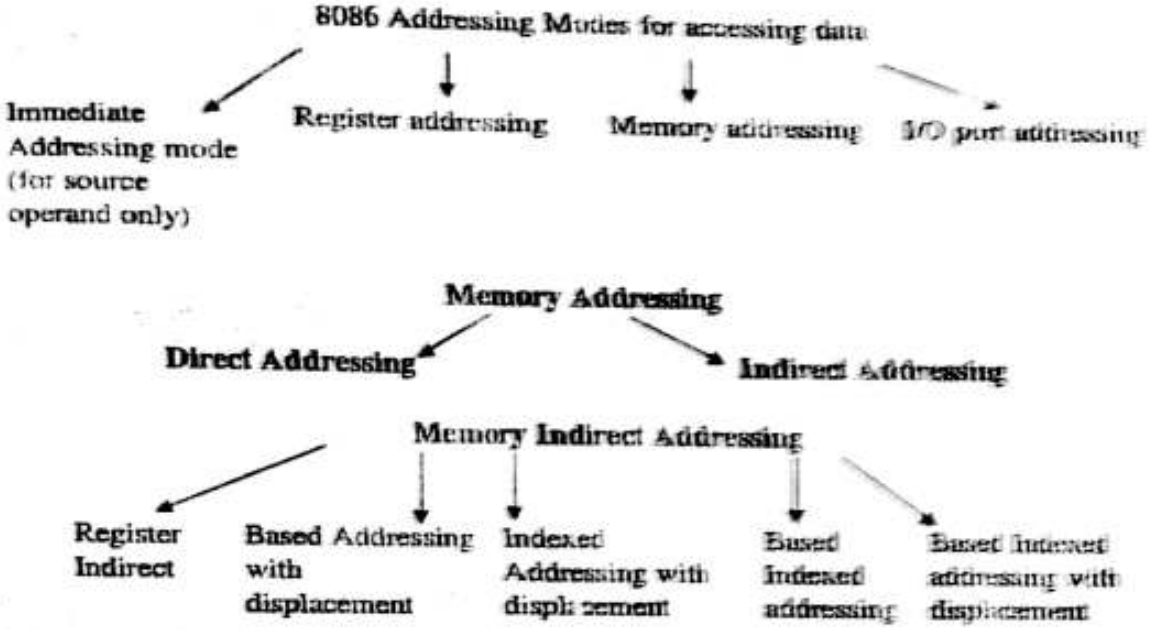
**Memory modes as derivatives of Based Indexed Addressing with Displacement**

| Instruction | Base Register | Index Register | Displacement | Addressing mode |
|---|---|---|---|---|
| MOV BX, DS:5634H | No | No | Yes | Direct Addressing |
| MOV CL, [SI] | No | Yes | No | Register Indirect |
| MOV DX, [BX] | Yes | No | No | |
| MOV DH, 2345H[BX] | Yes | No | Yes | Based Addressing with Displacement |
| MOV DX, 35H[DI] | No | Yes | Yes | Indexed Addressing with displacement |
| MOV CL, [SI+BX] | Yes | Yes | No | Based Indexed Addressing |
| MOV DL, 37H[BX+DI] | Yes | Yes | Yes | Based Indexed Addressing with displacement |



Scanned by CamScanner

# Summary of addressing mode




- **Implied:** the data value/data address is implicitly associated with the instruction.
- **Register** - references the data in a register or in a register pair.
- **Immediate**: the data is provided in the instruction.
- **Direct**: the instruction operand specifies the memory address where data is located.
- **Register indirect**: instruction specifies a register containing an address, where data is located. This addressing mode works with SI, DI, BX and BP registers.
- **Based**: 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.
- **Indexed**: 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.
- **Based Indexed**: the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.
- **Based Indexed with displacement**: 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.



Scanned by CamScanner

## I/O Ports

The simplest form of I/O interface is an I/O port. The data transfer between microprocessor and input device is done with the help of input port. The data transfer between microprocessor and output device is done with the help of output port.

**Input port :**

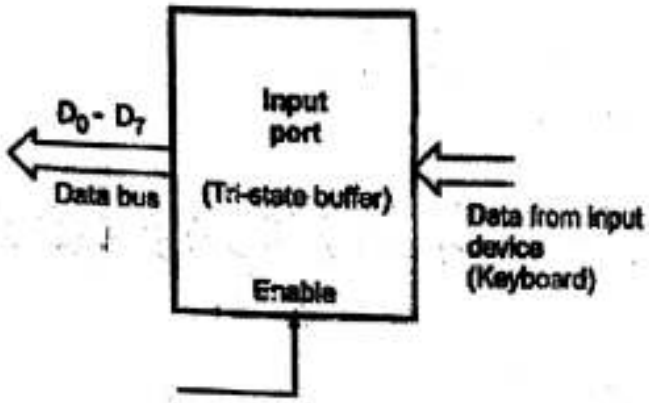



Fig. 1

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer as shown in the Fig. 1 . This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

**Output port :**

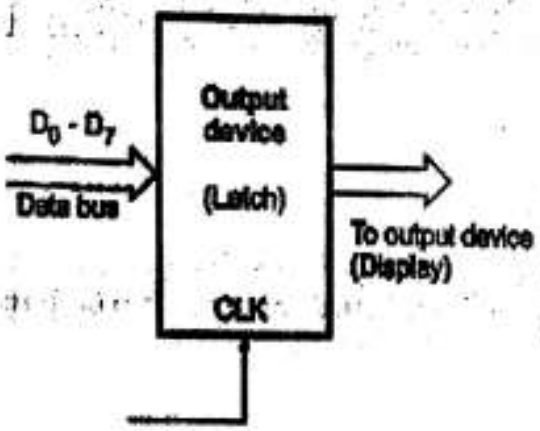
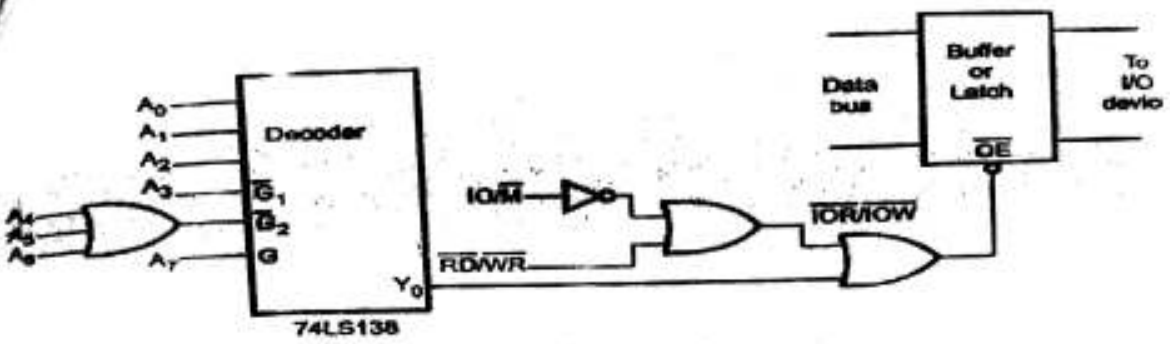



Fig. 2

It is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch, as shown in the Fig. 2 . When microprocessor wants to send data to the output device, it puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.



Scanned by CamScanner




. Absolute decoding circuit for the I/O device

**I/O port Addressing :**

I/O port Addressing

- Fixed port addressing Or Direct Port addressing
- Variable port addressing Or Indirect port addressing

**Fixed Port Addressing**

Ex. 1: IN AL, 83H

| | Before | After |
|---|---|---|
| AL | ~~34H~~ | 78H |
| Input port no. 83H | 78H | |

Ex. 2: IN AX, 83H

| | Before | After |
|---|---|---|
| AX | ~~5634H~~ | F278H |
| Input port no. 83H | 78H | |
| Input port no. 84H | F2H | |

Ex. 3: OUT 83H, AL

| | Before | After |
|---|---|---|
| AL | 50H | |
| Output port no. 83H | ~~65H~~ | 50H |

Ex. 4: OUT 83H, AX

| | Before | After |
|---|---|---|
| AX | 6050H | |
| Output port no. 83H | ~~65H~~ | 50H |
| Output port no. 84H | ~~40H~~ | 60H |

**IN and OUT instructions are allowed to use only AL or AX registers. Port address in the range 00 to FFH is provided in the instruction directly.**



Scanned by CamScanner

**Variable Port Addressing**

I/O port address is provided in DX register. Port address ranges from 0000 to FFFFH. Data transfer with AL or AX only.

**Ex. 1: IN AL, DX**

| | Before | After |
|---|---|---|
| AL | ~~30H~~ | 60H |
| DX | 1234H | |
| Input port no. 1234H | 60H | |

**Ex. 2: IN AX, DX**

| | Before | After |
|---|---|---|
| AX | ~~3040H~~ | 7060H |
| DX | 4000H | |
| Input port no. 4000H | 60H | |
| Input port no. 4001H | 70H | |

**Ex. 3: OUT DX, AL**

| | Before | After |
|---|---|---|
| AL | 65H | |
| DX | 5000H | |
| Output port no. 5000H | ~~80H~~ | 65H |

**Ex. 4: OUT DX, AX**

| | Before | After |
|---|---|---|
| AX | 4567H | |
| DX | 5000H | |
| Output port no. 5000H | ~~25H~~ | 67H |
| Output port no. 5001H | ~~36H~~ | 45H |

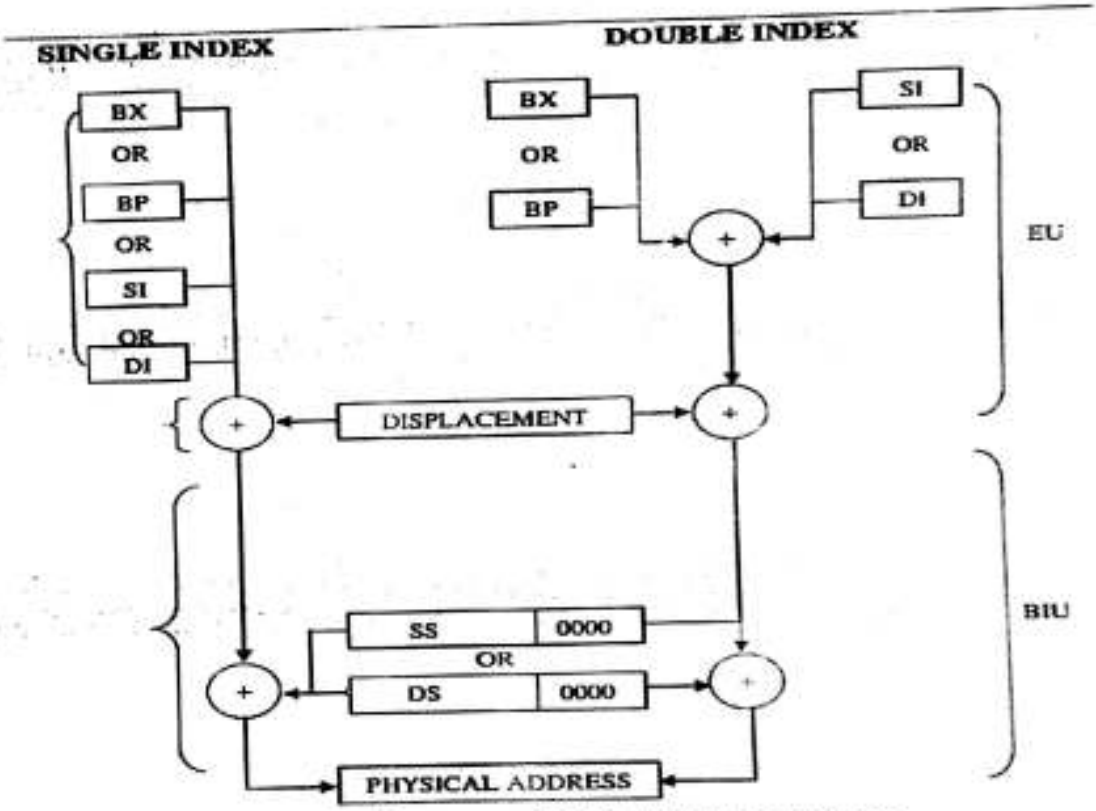
- IN & out transfer data between AL, AX and an external I/O device. The address of the I/O device is either stored with the instruction (fixed port) or in register DX (Variable port



Scanned by CamScanner

# Addressing Modes

| *Addressing Modes* | *Examples* |
|---|---|
| ❑ Immediate addressing | MOV AL, 12H |
| ❑ Register addressing | MOV AL, BL |
| ❑ Direct addressing | MOV [500H], AL |
| ❑ Register Indirect addressing | MOV DL, [SI] |
| ❑ Based addressing | MOV AX, [BX+4] |
| ❑ Indexed addressing | MOV [DI-8], BL |
| ❑ Based indexed addressing | MOV [BP+SI], AH |
| ❑ Based indexed with displacement addressing | MOV CL, [BX+DI+2] |




**Summary of 8086 Addressing Modes**



Scanned by CamScanner

| TYPE | INSTRUCTION | SOURCE | ADDRESS GENERATION | DESTINATION |
|---|---|---|---|---|
| 1) REGISTER | MOV AX, BX | REGISTER BX | | REGISTER AX |
| 2) IMMEDIATE | MOV CH, 3AH | DATA 3AH | | REGISTER CH |
| 3) DIRECT | MOV [1234], AX | REGISTER AX | (DS x 10H) + DISPLACEMENT<br>10000H + 1234 | MEMORY 11234H |
| 4) REGISTER INDIRECT | MOV [BX], CL | REGISTER CL | (DS x10H) + BX<br>10000H + 0300H | MEMORY 10300H |
| 5) BASE PLUS INDEX | MOV [BX + SI], BP | REGISTER BP | (DS x 10H) + BX + SI<br>10000H + 0300H + 0200H | MEMORY 10500H |
| 6) REGISTER RELATIVE | MOV CL, [BX +4] | MEMORY 10304H | (DS x 10H) + BX + 4<br>10000H + 0300H + 4 | REGISTER CL |
| 7) BASE RELATIVE PLUS INDEX | MOV ARRAY [BX + SI], DX | REGISTER DX | (DS x 10H) + ARRAY + BX + SI<br>10000H+1000H+0300H+0200H | MEMORY 11500H |

ASSUME: BX = 0300H; SI = 0200H; ARRAY = 1000H; DS = 1000H



Scanned by CamScanner

DESTINATION REGISTER AX

## 16 bit Segment Register Assignments

| Type of Memory Reference | Default Segment | Alternate Segment | Offset |
|---|---|---|---|
| Instruction Fetch | CS | none | IP |
| Stack Operations | SS | none | SP,BP |
| General Data | DS | CS,ES,SS | BX, address |
| String Source | DS | CS,ES,SS | SI, DI, address |
| String Destination | ES | None | DI |

## Segment override

| Segment Register | CS | DS | ES | SS |
|---|---|---|---|---|
| Offset Register | IP | SI,DI,BX | SI,DI,BX | SP,BP |

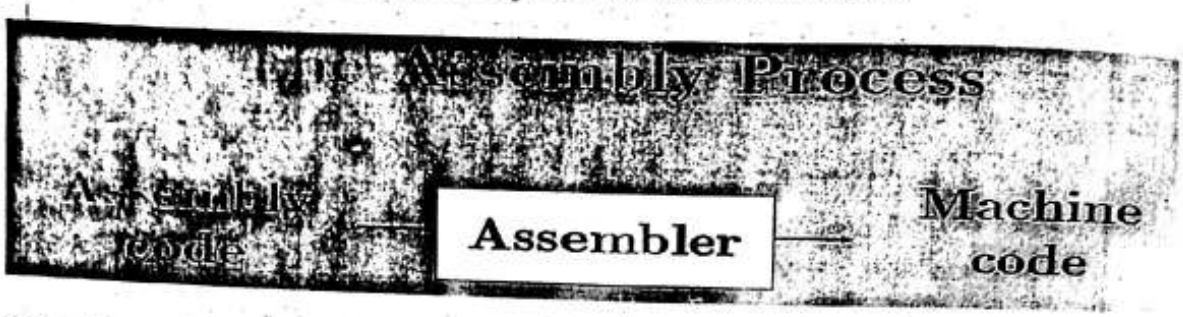
| Instruction Examples | Override Segment Used | Default Segment |
|---|---|---|
| MOV AX,CS:[BP] | CS:BP | SS:BP |
| MOV DX,SS:[SI] | SS:SI | DS:SI |
| MOV AX,DS:[BP] | DS:BP | SS:BP |
| MOV CX,ES:[BX+12] | ES:BX+12 | DS:BX+12 |
| MOV SS:[BX][DI+32],AX | SS:BX+DI+32 | DS:BX+DI+32 |



Scanned by CamScanner

# Converting Assembly language Instruction to Machine code

**Assembler** translates assembly code to machine code




**What is an assembly language?**

Assembly language is a low level programming language. You need to get some knowledge about computer structure in order to understand anything.

**Memory Access**

To access memory we can use these four registers: **BX, SI, DI, BP.** Combining these registers inside [ ] symbols, we can get different memory locations. These combinations are supported (addressing modes):

| | | |
|---|---|---|
| [BX + SI]<br>[BX + DI]<br>[BP + SI]<br>[BP + DI] | [SI]<br>[DI]<br>d16 (variable offset only)<br>[BX] | [BX + SI] + d8<br>[BX + DI] + d8<br>[BP + SI] + d8<br>[BP + DI] + d8 |
| [SI] + d8<br>[DI] + d8<br>[BP] + d8<br>[BX] + d8 | [BX + SI] + d16<br>[BX + DI] + d16<br>[BP + SI] + d16<br>[BP + DI] + d16 | [SI] + d16<br>[DI] + d16<br>[BP] + d16<br>[BX] + d16 |

- **d8** - stays for 8 bit displacement

- **d16** - stays for 16 bit displacement.



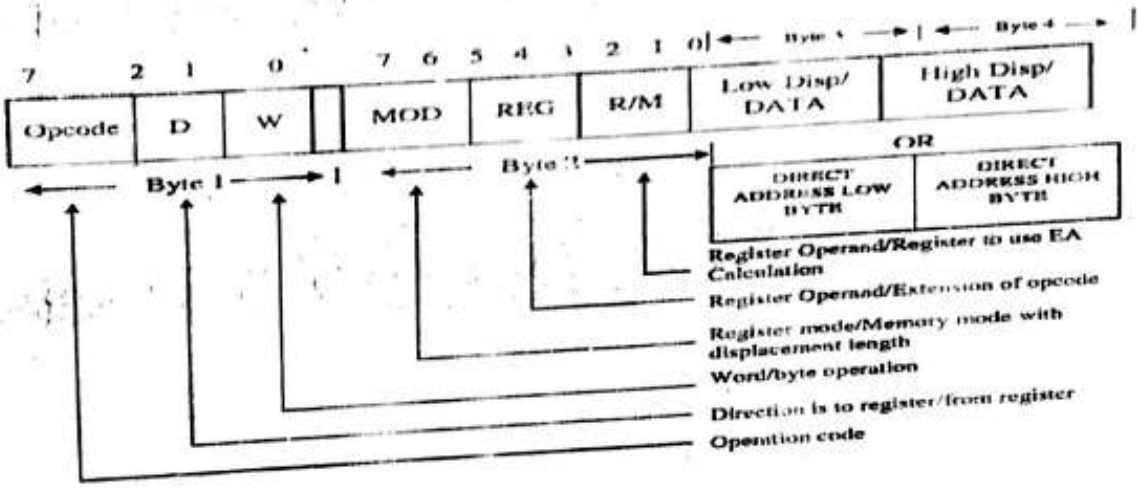
Scanned by CamScanner

- Displacement can be inside or outside of [ ] symbols, compiler generates the same machine code for both ways.

- Displacement is a signed value, so it can be both positive or negative. Generally the compiler takes care about difference between d8 and d16, and generates the required machine code.

## Instruction Format

- 8086 Instructions are represented as binary numbers Instructions require between 1 and 6 bytes.

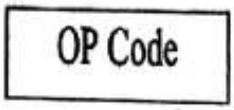



| byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | opcode | | | | | | d | w | Opcode byte |
| 2 | mod | | reg | | | r/m | | | Addressing mode byte |
| 3 | [optional] | | | | | | | | low disp, addr, or data |
| 4 | [optional] | | | | | | | | high disp, addr, or data |
| 5 | [optional] | | | | | | | | low data |
| 6 | [optional] | | | | | | | | high data |

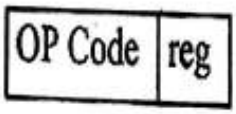


Scanned by CamScanner

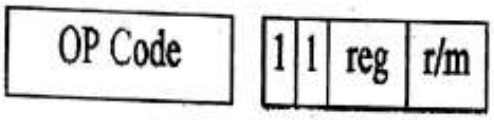
1. One-Byte Instruction (Implied Operand)



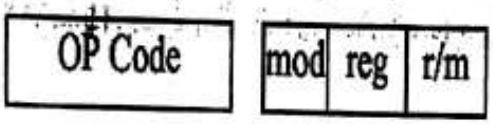

2. One-Byte Instruction (Register Mode)



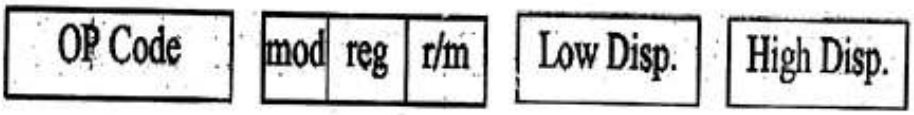

3. Register to Register



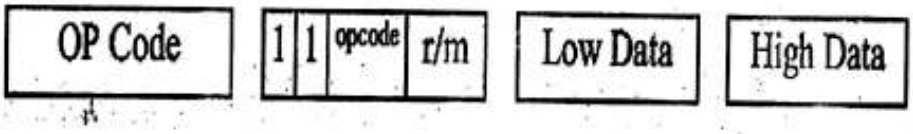

4. Register to/from memory with no displacement



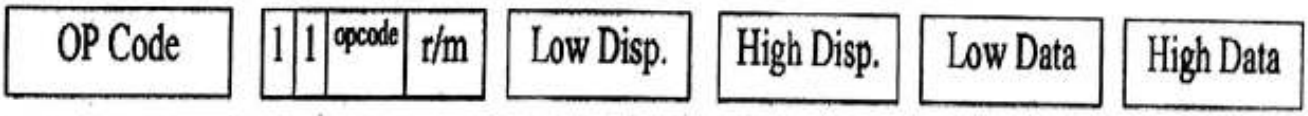

5. Register to/from memory with displacement




6. Immediate operand to Register



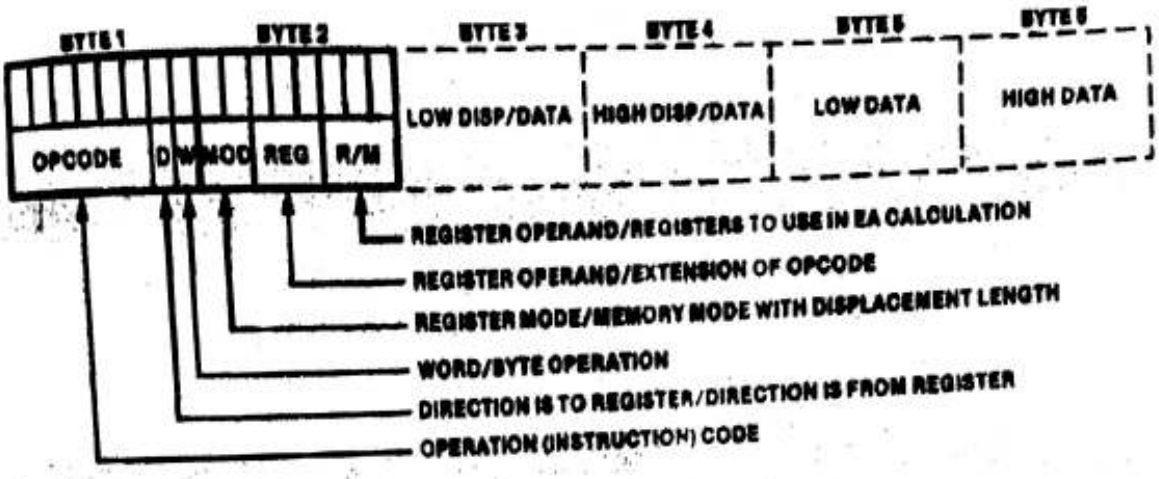

7. Immediate operand to Memory with 16-bit displacement






Scanned by CamScanner

# General Instruction Format




## 1- Byte 1 information:

- **Opcode f ield (6-bits)**—specifies the operation to be performed by the instruction
- **D (1-bit)**—register direction: tells whether the register which is selected by the **REG** field in the second byte is the source or destination

**D = 0 :** source operand

**D = 1 :** destination operand

- **W (1-bit)**—data size word/byte for all registers

**W = 0 :** indicates 8 bit operation

**W = 1 :** indicates 16 bit operation

**The S, V, Z fields of the opcode in specific instructions**

| Field | Value | Function |
|---|---|---|
| S | 0 | No sign extension |
| | 1 | Sign extend 8-bit immediate data to 16 bits if W=1 |
| V | 0 | Shift/rotate count is one |
| | 1 | Shift/rotate count is specified in CL register |
| Z | 0 | Repeat/loop while zero flag is clear |
| | 1 | Repeat/loop while zero flag is set |



Scanned by CamScanner

## 2- Byte 2 information

The second byte of the instruction usually identifies whether one of the operands is in memory or whether both are registers.

This byte contains **3 fields**. These are the mode (MOD) , field, the register (REG), field and the Register/Memory (R/M)

### MOD (2-bit mode field):

Specifies the type of the second operand

| MOD (2 bits) | Interpretation |
|---|---|
| 00 | Memory mode with no displacement follows except for 16 bit displacement when R/M=110 |
| 01 | Memory mode with 8 bit displacement |
| 10 | Memory mode with 16 bit displacement |
| 11 | Register mode (no displacement) |

### REG (3-bit register field):

Register field occupies 3 bits. It defines the register for the first operand which is specified as source or destination by the D bit.

| REG | W=0 | W=1 |
|---|---|---|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

| reg Value | Segment Registers |
|---|---|
| 000 | ES |
| 001 | CS |
| 010 | SS |
| 011 | DS |



Scanned by CamScanner

## R/M (3-bit register/memory field)–

The R/M field occupies 3 bits. The R/M field along with the MOD field defines the second operand as a register or a storage location in memory as shown below.

**MOD 11**

| R/M | W=0 | W=1 |
|---|---|---|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

**Effective Address Calculation**

| R/M | MOD=00 | MOD 01 | MOD 10 |
|---|---|---|---|
| 000 | (BX) + (SI) | (BX)+(SI)+D8 | (BX)+(SI)+D16 |
| 001 | (BX)+(DI) | (BX)+(DI)+D8 | (BX)+(DI)+D16 |
| 010 | (BP)+(SI) | (BP)+(SI)+D8 | (BP)+(SI)+D16 |
| 011 | (BP)+(DI) | (BP)+(DI)+D8 | (BP)+(DI)+D10 |
| 100 | (SI) | (SI) + D8 | (SI) + D16 |
| 101 | (DI) | (DI) + D8 | (DI) + D16 |
| 110 | Direct address | (BP) + D8 | (BP) + D16 |
| 111 | (BX) | (BX) + D8 | (BX) + D16 |



Scanned by CamScanner

## Example

- MOV AL,BL
- Opcode for MOV = 100010
- We'll encode AL so
  D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

| OPCODE | D | W | MOD | REG | R/M |
|---|---|---|---|---|---|
| 100010 | 0 | 0 | 11 | 000 | 011 |

MOV BL,AL => 10001000 11000011 = 88 C3h

**Example** : MOV CH, BL
This instruction transfers 8 bit content of BL

**Into CH**
The 6 bit Opcode for this instruction is $100010_2$ D bit indicates whether the register specified by the REG field of byte 2 is a source or destination operand.
D=0 indicates BL is a source operand.
W=0 byte operation
In byte 2, since the second operand is a register MOD field is $11_2$.
The R/M field = 101 (CH)
Register (REG) field = 011 (BL)
Hence the machine code for MOV CH, BL is
10001000 11 011 101
Byte 1 Byte2
= 88DD16

**Example :**
Give the instruction template and generate the code for the instruction

MOV AX , [BX]

(Code for MOV instruction is 100010)
AX destination register with D=1 and code for AX is 000 [BX] is specified using 00
Mode and R/M value 111
It is a word operation

| Opcode | D | W | Mod | REG | R/M | |
|---|---|---|---|---|---|---|
| 100010 | 1 | 1 | 00 | 000 | 111 | =8B 07H |

**Example . : Code for MOV 1234 (BP), DX**

Here we have specify DX using REG field, the D bit must be 0, indicating the DX is the source register. The REG field must be 010 to indicate DX register. The W bit must be 1 to indicate it is a word operation. 1234 [BP] is specified using MOD value of 10 and R/M value of 110 and a displacement of 1234H. The 4 byte code for this instruction would be 89 96 34 12H.

| Opcode | D | W | MOD | REG | R/M | LB displacement | HB displacement |
|---|---|---|---|---|---|---|---|
| 100010 | 0 | 1 | 10 | 010 | 110 | 34H | 12H |



Scanned by CamScanner

Register/Memory to/from Register

| 100010DW | MOD | REG | R/M |
|---|---|---|---|

V DL, [DI]

p code MOV

D: Transfer to Register REG

W: byte

MOD: No displacement

R/M: DS:[DI]

| 100010 | D 1 | W 0 | MOD 00 | REG 010 | R/M 101 |
|---|---|---|---|---|---|

8A 15

---

② MOV BP, SP

| | D | W | MOD | REG | R/M |
|---|---|---|---|---|---|
| 100010 | 1 | 1 | 11 | 101 (BP) | 100 (SP) |

8BEC

---

Immediate to Register/Memory

③

| 1100011W | MOD | 000 | r/M | data Low | data Hi2 |
|---|---|---|---|---|---|

MOV [BX], 3322

BX

1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1

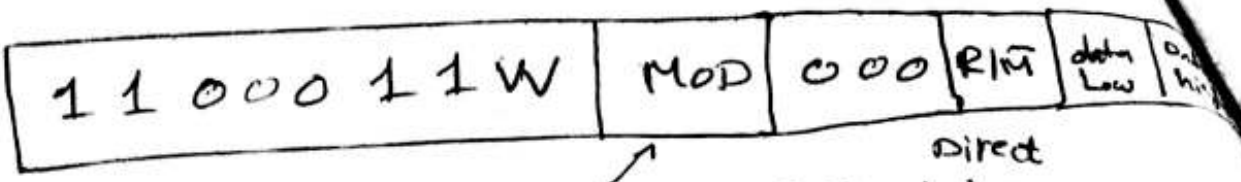
C7 07 2 2 3 3

C7
07
22
33

108

Scanned by CamScanner

(4) Mov [1000h], 3322h

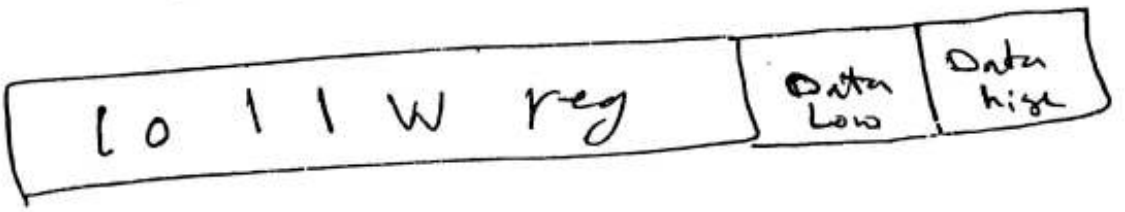



1 1 0 0 0 1 1 1 | 0 0 | 0 0 0 | 1 1 0
C 7 | 0 | 0 | 6

C7
06
00 } Address
10 }
22 } Data
33 }

6-byte

(5) Imm. to Reg




Mov AX, 5566h

W AX 6 6 5 5
1 0 1 1 | 1 | 0 0 0 | 0110 0110 0101 0101

B8
66
55



Scanned by CamScanner

DD AX, 7080H

Imm. to Accumulator

| 0 0 0 0 0 1 0 W | data | data if w=1 |
|---|---|---|

ADD AX, 7080H

W

0 0 0 0 0 1 0 1 | 1 0 0 0 | 0 0 0 0 | 0 1 1 1 | 0 0 0 0

0 5 | 8 0 | 7 0

05
80
70

---

* INC AX

| 0 1 0 0 0 reg |
|---|

AX

0 1 0 0 0 0 0 0

40

---

INC b.[BX]

1 1 1 1 1 1 1 W mod 0 0 0 r/m

1 1 1 1 | 1 1 1 0 | 0 0 0 0 0 1 1 1

F E | 0 7

FE
07

INC W.[BX]

1 1 1 1 1 1 1 W mod 0 0 0 r/m

1 1 1 1 | 1 1 1 1 | 0 0 0 0 0 1 1 1

F F | 0 7

FF
07



Scanned by CamScanner

IN

① Fixed Port | 1 1 1 0 0 1 0 W | Port |

Ex IN AL, 03H

W 0 3

1 1 1 0 0 1 0 0 0000 0011

E 4

0 3

---

IN AX, 03H

1 1 1 0 0 1 0 1 0000 0011

E 5

0 3

---

② Variable Port | 1 1 1 0 1 1 0 W |

IN AL, DX

1 1 1 0 1 1 0 0

EC

IN AX, DX

1 1 1 0 1 1 0 1

E D

\|\|\|

Scanned by CamScanner

# HW#3

1- Although 8086 is a 16-bit MP, it deals with 8-bit memory. Why?

2- What is meant by addressing mode?

3- Name the different addressing modes of 8086?

4- Mention the different memory addressing modes?

5- How the physical address is generated for the different memory addressing modes?

6- Give examples each of (a) Register Addressing mode (b) Immediate Addressing Mode?

7- List the 16- bit registers used for register addressing?

8- The move instruction is placed in what field of a statement?

9- Discuss Based Addressing Mode?

10- Discuss Indexed Addressing Mode.

11- Discuss Indexed Addressing Mode.

12- Discuss Based Indexed Addressing mode.

13- Discuss Based Indexed with displacement Addressing Mode.

13- What do the following MOV instruction accomplish?

a- MOV Al,55H

b- MOV BL,[1000H]

c- MOV CX,[BX]

d- MOV CX,[BX+DI]

e- MOV AX,[DI+200]

15- Suppose that DS=1000H, SS=2000H, BP=1000H and DI=0100H. Determine address accessed by each of the following.

a- MOV Al,[BP+DI]

b- MOV CX,[DI]

c- MOV DX,[BP]



Scanned by CamScanner

intel.

## Table 2. Instruction Set Summary

| Mnemonic and Description | Instruction Code | | | |
|---|---|---|---|---|
| | 76543210 | 76543210 | 76543210 | 76543210 |
| **DATA TRANSFER** | | | | |
| **MOV = Move:** | | | | |
| Register/Memory to/from Register | 100010dw | mod reg r/m | | |
| Immediate to Register/Memory | 1100011w | mod 000 r/m | data | data if w = 1 |
| Immediate to Register | 1011wreg | data | data if w = 1 | |
| Memory to Accumulator | 1010000w | addr-low | addr-high | |
| Accumulator to Memory | 1010001w | addr-low | addr-high | |
| Register/Memory to Segment Register | 10001110 | mod 0 reg r/m | | |
| Segment Register to Register/Memory | 10001100 | mod 0 reg r/m | | |
| **PUSH = Push:** | | | | |
| Register/Memory | 11111111 | mod 110 r/m | | |
| Register | 01010reg | | | |
| Segment Register | 000reg110 | | | |
| **POP = Pop:** | | | | |
| Register/Memory | 10001111 | mod 000 r/m | | |
| Register | 01011reg | | | |
| Segment Register | 000reg111 | | | |
| **XCHG = Exchange:** | | | | |
| Register/Memory with Register | 1000011w | mod reg r/m | | |
| Register with Accumulator | 10010reg | | | |
| **IN = Input from:** | | | | |
| Fixed Port | 1110010w | port | | |
| Variable Port | 1110110w | | | |
| **OUT = Output to:** | | | | |
| Fixed Port | 1110011w | port | | |
| Variable Port | 1110111w | | | |
| **XLAT = Translate Byte to AL** | 11010111 | | | |
| **LEA = Load EA to Register** | 10001101 | mod reg r/m | | |
| **LDS = Load Pointer to DS** | 11000101 | mod reg r/m | | |
| **LES = Load Pointer to ES** | 11000100 | mod reg r/m | | |
| **LAHF = Load AH with Flags** | 10011111 | | | |
| **SAHF = Store AH into Flags** | 10011110 | | | |
| **PUSHF = Push Flags** | 10011100 | | | |
| **POPF = Pop Flags** | 10011101 | | | |

Mnemonics © Intel, 1978



113

Scanned by CamScanner

**Table 2. Instruction Set Summary (Continued)**

| Mnemonic and Description | Instruction Code | | | |
|---|---|---|---|---|
| **ARITHMETIC** | 76543210 | 76543210 | 76543210 | 76543210 |
| **ADD = Add:** | | | | |
| Reg./Memory with Register to Either | 000000dw | mod reg r/m | | |
| Immediate to Register/Memory | 100000sw | mod 000 r/m | data | data if s:w = 01 |
| Immediate to Accumulator | 0000010w | data | data if w = 1 | |
| **ADC = Add with Carry:** | | | | |
| Reg./Memory with Register to Either | 000100dw | mod reg r/m | | |
| Immediate to Register/Memory | 100000sw | mod 010 r/m | data | data if s:w = 01 |
| Immediate to Accumulator | 0001010w | data | data if w = 1 | |
| **INC = Increment:** | | | | |
| Register/Memory | 1111111w | mod 000 r/m | | |
| Register | 01000reg | | | |
| **AAA = ASCII Adjust for Add** | 00110111 | | | |
| **BAA = Decimal Adjust for Add** | 00100111 | | | |
| **SUB = Subtract:** | | | | |
| Reg./Memory and Register to Either | 001010dw | mod reg r/m | | |
| Immediate from Register/Memory | 100000sw | mod 101 r/m | data | data if s:w = 01 |
| Immediate from Accumulator | 0010110w | data | data if w = 1 | |
| **SSB = Subtract with Borrow** | | | | |
| Reg./Memory and Register to Either | 000110dw | mod reg r/m | | |
| Immediate from Register/Memory | 100000sw | mod 011 r/m | data | data if s:w = 01 |
| Immediate from Accumulator | 0001110w | data | data if w = 1 | |
| **DEC = Decrement:** | | | | |
| Register/memory | 1111111w | mod 001 r/m | | |
| Register | 01001 reg | | | |
| **NEG = Change sign** | 1111011w | mod 011 r/m | | |
| **CMP = Compare:** | | | | |
| Register/Memory and Register | 001110dw | mod reg r/m | | |
| Immediate with Register/Memory | 100000sw | mod 111 r/m | data | data if s:w = 01 |
| Immediate with Accumulator | 0011110w | data | data if w = 1 | |
| **AAS = ASCII Adjust for Subtract** | 00111111 | | | |
| **DAS = Decimal Adjust for Subtract** | 00101111 | | | |
| **MUL = Multiply (Unsigned)** | 1111011w | mod 100 r/m | | |
| **IMUL = Integer Multiply (Signed)** | 1111011w | mod 101 r/m | | |
| **AAM = ASCII Adjust for Multiply** | 11010100 | 00001010 | | |
| **DIV = Divide (Unsigned)** | 1111011w | mod 110 r/m | | |
| **IDIV = Integer Divide (Signed)** | 1111011w | mod 111 r/m | | |
| **AAD = ASCII Adjust for Divide** | 11010101 | 00001010 | | |
| **CBW = Convert Byte to Word** | 10011000 | | | |
| **CWD = Convert Word to Double Word** | 10011001 | | | |

Mnemonics © Intel, 1978



114

Scanned by CamScanner

## Table 2. Instruction Set Summary (continued)

| Mnemonic and Description | Instruction Code | | | |
|---|---|---|---|---|
| | 76543210 | 76543210 | 76543210 | 76543210 |
| **LOGIC** | | | | |
| **NOT** = Invert | 1111011w | mod 0 1 0 r/m | | |
| **SHL/SAL** = Shift Logical/Arithmetic Left | 110100vw | mod 1 0 0 r/m | | |
| **SHR** = Shift Logical Right | 110100vw | mod 1 0 1 r/m | | |
| **SAR** = Shift Arithmetic Right | 110100vw | mod 1 1 1 r/m | | |
| **ROL** = Rotate Left | 110100vw | mod 0 0 0 r/m | | |
| **ROR** = Rotate Right | 110100vw | mod 0 0 1 r/m | | |
| **RCL** = Rotate Through Carry Flag Left | 110100vw | mod 0 1 0 r/m | | |
| **RCR** = Rotate Through Carry Right | 110100vw | mod 0 1 1 r/m | | |
| **AND** = And | | | | |
| Reg./Memory and Register to Either | 001000dw | mod reg r/m | | |
| Immediate to Register/Memory | 1000000w | mod 1 0 0 r/m | data | data if w = 1 |
| Immediate to Accumulator | 0010010w | data | data if w = 1 | |
| **TEST** = And Function to Flags, No Result | | | | |
| Register/Memory and Register | 1000010w | mod reg r/m | | |
| Immediate Data and Register/Memory | 1111011w | mod 0 0 0 r/m | data | data if w = 1 |
| Immediate Data and Accumulator | 1010100w | data | data if w = 1 | |
| **OR** = Or | | | | |
| Reg./Memory and Register to Either | 000010dw | mod reg r/m | | |
| Immediate to Register/Memory | 1000000w | mod 0 0 1 r/m | data | data if w = 1 |
| Immediate to Accumulator | 0000110w | data | data if w = 1 | |
| **XOR** = Exclusive or | | | | |
| Reg./Memory and Register to Either | 001100dw | mod reg r/m | | |
| Immediate to Register/Memory | 1000000w | mod 1 1 0 r/m | data | data if w = 1 |
| Immediate to Accumulator | 0011010w | data | data if w = 1 | |
| **STRING MANIPULATION** | | | | |
| **REP** = Repeat | 1111001z | | | |
| **MOVS** = Move Byte/Word | 1010010w | | | |
| **CMPS** = Compare Byte/Word | 1010011w | | | |
| **SCAS** = Scan Byte/Word | 1010111w | | | |
| **LODS** = Load Byte/Wd to AL/AX | 1010110w | | | |
| **STOS** = Stor Byte/Wd from AL/A | 1010101w | | | |
| **CONTROL TRANSFER** | | | | |
| **CALL** = Call | | | | |
| Direct within Segment | 11101000 | disp-low | disp-high | |
| Indirect within Segment | 11111111 | mod 0 1 0 r/m | | |
| Direct Intersegment | 10011010 | offset-low | offset-high | |
| | | seg-low | seg-high | |
| Indirect Intersegment | 11111111 | mod 0 1 1 r/m | | |

Mnemonics © Intel, 1978



115

Scanned by CamScanner

**Table 2. Instruction Set Summary** (Continued)

| Mnemonic and Description | Instruction Code | | | |
|---|---|---|---|---|
| **JMP** Unconditional Jump: | 76543210 | 76543210 | 76543210 | |
| Direct within Segment | 11101001 | disp-low | disp-high | |
| Direct within Segment-Short | 11101011 | disp | | |
| Indirect within Segment | 11111111 | mod 100 r/m | | |
| Direct Intersegment | 11101010 | offset-low | offset-high | |
| | | seg-low | seg-high | |
| Indirect Intersegment | 11111111 | mod 101 r/m | | |
| **RET** = Return from CALL: | | | | |
| Within Segment | 11000011 | | | |
| Within Seg Adding Immed to SP | 11000010 | data-low | data high | |
| Intersegment | 11001011 | | | |
| Intersegment Adding Immediate to SP | 11001010 | data-low | data-high | |
| **JE/JZ** = Jump on Equal/Zero | 01110100 | disp | | |
| **JL/JNGE** = Jump on Less/Not Greater or Equal | 01111100 | disp | | |
| **JLE/JNG** = Jump on Less or Equal/ Not Greater | 01111110 | disp | | |
| **JB/JNAE** = Jump on Below/Not Above or Equal | 01110010 | disp | | |
| **JBE/JNA** = Jump on Below or Equal/ Not Above | 01110110 | disp | | |
| **JP/JPE** = Jump on Parity/Parity Even | 01111010 | disp | | |
| **JO** = Jump on Overflow | 01110000 | disp | | |
| **JS** = Jump on Sign | 01111000 | disp | | |
| **JNE/JNZ** = Jump on Not Equal/Not Zero | 01110101 | disp | | |
| **JNL/JGE** = Jump on Not Less/Greater or Equal | 01111101 | disp | | |
| **JNLE/JG** = Jump on Not Less or Equal/ Greater | 01111111 | disp | | |
| **JNB/JAE** = Jump on Not Below/Above or Equal | 01110011 | disp | | |
| **JNBE/JA** = Jump on Not Below or Equal/Above | 01110111 | disp | | |
| **JNP/JPO** = Jump on Not Par/Par Odd | 01111011 | disp | | |
| **JNO** = Jump on Not Overflow | 01110001 | disp | | |
| **JNS** = Jump on Not Sign | 01111001 | disp | | |
| **LOOP** = Loop CX Times | 11100010 | disp | | |
| **LOOPZ/LOOPE** = Loop While Zero/Equal | 11100001 | disp | | |
| **LOOPNZ/LOOPNE** = Loop While Not Zero/Equal | 11100000 | disp | | |
| **JCXZ** = Jump on CX Zero | 11100011 | disp | | |
| **INT** = Interrupt | | | | |
| Type Specified | 11001101 | type | | |
| Type 3 | 11001100 | | | |
| **INTO** = Interrupt on Overflow | 11001110 | | | |
| **IRET** = Interrupt Return | 11001111 | | | |



16

Scanned by CamScanner

intel.

**Table 2. Instruction Set Summary (Continued)**

| Mnemonic and Description | Instruction Code 76543210 | 76543210 |
|---|---|---|
| **PROCESSOR CONTROL** | | |
| CLC = Clear Carry | 11111000 | |
| CMC = Complement Carry | 11110101 | |
| STC = Set Carry | 11111001 | |
| CLD = Clear Direction | 11111100 | |
| STD = Set Direction | 11111101 | |
| CLI = Clear Interrupt | 11111010 | |
| STI = Set Interrupt | 11111011 | |
| HLT = Halt | 11110100 | |
| WAIT = Wait | 10011011 | |
| ESC = Escape (to External Device) | 11011xxx | mod x x x r/m |
| LOCK = Bus Lock Prefix | 11110000 | |

NOTES:
AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction
if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
if mod = 10 then DISP = disp-high; disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)
*except if mod = 00 and r/m = 110 then EA = disp-high; disp-low.

Mnemonics © Intel, 1978

if s w = 01 then 16 bits of immediate data form the operand
if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG

**SEGMENT OVERRIDE PREFIX**

| 0 0 1 reg 1 1 0 |
|---|

REG is assigned according to the following table:

| 16-Bit (w = 1) | 8-Bit (w = 0) | Segment |
|---|---|---|
| 000 AX | 000 AL | 00 ES |
| 001 CX | 001 CL | 01 CS |
| 010 DX | 010 DL | 10 SS |
| 011 BX | 011 BL | 11 DS |
| 100 SP | 100 AH | |
| 101 BP | 101 CH | |
| 110 SI | 110 DH | |
| 111 DI | 111 BH | |

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:
FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

## DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -004 data sheet. Please review this summary carefully.

1. The Intel 8086 implementation technology (HMOS) has been changed to (HMOS-III).
2. Delete all "changes from 1985 Handbook Specification" sentences.



Scanned by CamScanner

Questions & answers Part 1

# Microprocessor, Microcomputer and Associated Languages

**1. What is meant by distributed processing?**
**Ans.** Distributed processing involves the use of several microprocessors in a single computer system. For example, for such a system, the first microprocessor may control keyboard activities, the second controls storage devices like disk drives, the third controls input/output operations, while the fourth may act as the main system processor.

**2. When was the first microprocessor developed?**
**Ans.** The first microprocessor was developed by BUSICOM of Japan and INTEL of USA in the year 1971.

**3. What is a microprocessor?**
**Ans.** A microprocessor may be thought of as a silicon chip around which a microcomputer is built.

**4. What is the technology used in microprocessors?**
**Ans.** NMOS technology is used in microprocessors.

**5. What are the three main units of a digital computer?**
**Ans.** The three main units of a digital computer are: the central processing unit (CPU), the memory unit and the input/output devices.

**6. How does the microprocessor communicate with the memory and input/output devices?**
**Ans.** The microprocessor communicates with the memory and the Input/Output devices via the three buses, viz., data bus, address bus and control bus.

**7. What are the different jobs that the CPU is expected to do at any given point of time?**
**Ans.** The CPU may perform a memory read or write operation, an I/O read or write operation or an internal activity.

**9. What is a mnemonic?**
**Ans.** It is very difficult to understand a program if it is written in either binary or hex code. Thus the manufacturers have devised a symbolic code for each instruction, called a mnemonic. Examples of mnemonics are: INR A, ADD M, etc.
3C 86

**10. What is machine language programming?**
**Ans.** Programming a computer by utilising hex or binary code is known as machine language programming.

**11. What is meant by assembly language programming?**
**Ans.** Programming a microcomputer by writing mnemonics is known as assembly language programming.



Scanned by CamScanner

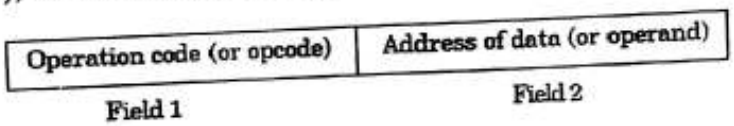
**12. What are meant by low level and high level languages?**
**Ans.** Programming languages that are machine dependent are called low level languages. For example, assembly language is a low level language.
On the other hand, programming languages that are machine independent are called high level languages. Examples are BASIC, FORTRAN, C, ALGOL, COBOL, etc.

**13. What is meant by 'word length' of a computer?**
**Ans.** The number of bits that a computer recognizes and can process at a time is known as its 'word length'.

**14. What is meant by instruction?**
**Ans.** An instruction is a command which asks the microprocessor to perform a specific task or job.

**15. How many different instructions mP 8085 has? What is an instruction set?**
**Ans.** 8085 microprocessor has a total of 74 different instructions for performing different operations or tasks. The entire different instructions that a particular microprocessor can handle is called i s instruction set.

**16. What an instruction consists of?**
**Ans.** An instruction consists of an operation code (called 'opcode') and the address of the data (called 'operand'), on which the opcode operates.




**17. Give one example each of the different types of instructions.**
**Ans.** Instructions can be of (i) direct (ii) immediate (iii) implicit type. Examples of each type follows:
(i) direct type : LDA 4000
(ii) immediate type : MVIA, 1F
(iii) implicit type : ADD C

**18. What language a microprocessor understands?**
**Ans.** Microprocessor understands only binary language.

**19. How the mnemonics written in assembly language are translated into binary?**
**Ans.** The translation from assembly language (i.e., mnemonics) into binary is done either manually (known as hand (or manual) assembly) or by a program called an assembler.

Thus an assembler can be thought of as a program which translates the mnemonics entered by an ASCII keyboard into its equivalent binary code, which is the only one understood by a microprocessor.



Scanned by CamScanner

**20. How an assembler translates programs written in mnemonic form to binary?**
Ans. An assembler has a 'translation dictionary', which is stored in its memory. Mnemonics entered via keyboard is compared with this dictionary, which then retrieves its binary equivalent from the same place (dictionary).

**21. What are the types of mnemonics possible?**
Ans. Mnemonics can be of two types—alphabetical or alphanumerical.
Example of first type is ADD D whereas, MVI A, 0F is an example of the second category.

**22. What are source codes and object codes?**
Ans. High level languages (like, BASIC, COBOL, ALGOL, etc.) are called source codes, whereas binary language is called object code.

**23. How are high level languages converted into binary?**
Ans. The high level languages are converted into their corresponding binary by means of another program, called either a compiler or an interpreter.
Compilers are generally used for FORTAN, PASCAL, COBOL, etc. whereas interpreters are generally used in BASIC. M-BASIC is a common example of an interpreter for BASIC language.

**24. What is a 'statement'?**
Ans. Instructions written in high level languages are known as statements.

**25. Write down the difference between a compiler and an interpreter.**
Ans. Difference between a compiler and an interpreter lies in the generation of the machine code or object code.
A compiler reads the entire program first and then generates the corresponding object code. Whereas, an interpreter reads an instruction at a time, produces the corresponding object code and executes the same before it starts reading the next instruction. A program from a compiler runs some 5 to 25 times faster than a program from an interpreter.

**26. Differentiate between a compiler/interpreter and an assembler.**

| S.No. | Compiler/Interpreter | Assembler |
|---|---|---|
| 1. | Debugging easier | Debugging relatively tougher |
| 2. | Less efficient | More efficient |
| 3. | Requires large memory space | Requires less memory space |

**27. What are the names given to instructions written in high and low level languages?**
Ans. The instructions written in high level languages are known as 'statements' whereas those written in low level languages are called 'mnemonics'.



Scanned by CamScanner

**28. What is another name of a microprocessor?**
Ans. A microprocessor is also called the CPU, the central processing unit.

**29. What is a microcomputer?**
Ans. A microcomputer is a system which is capable of processing a stream of input information's and will generate a stream of output information's taking the help of a program which is stored in the memory of the system.

**30. What are the jobs that a microcomputer is capable of doing? How does it do the jobs?**
Ans. A microcomputer is capable of doing the following jobs:
(a) receiving input (in the form of data or instruction).
(b) performing computations, both arithmetic and logical.
(c) storing data and instructions.
(d) displaying the results of any computations.
(e) controlling all the devices that perform the above mentioned four tasks, either directly or indirectly.
The first task is done by an input device, whereas the second job is done by the arithmetic logic unit (ALU). The third task is done by the memory unit while an output device does the fourth job. The fifth job is executed by the timing and control (T&C) unit.

**31. What is a bus?**
Ans. A bus is a bunch of wires through which data or address or control signals flow.

**32. What are the different buses and what jobs they do in a microprocessor?**
Ans. The different buses in a microprocessor are the data bus (DB), address bus (AB) and the control bus (CB). Data flow through the DB, while address comes out of the AB and CB controls the activities of the microcomputer system at any instant of time.

**33. Why are the different buses buffered?**
Ans. A buffer enhances the current or power driving capability of a pin of an IC.

**34. In how many ways computers are divided?**
Ans. Computers are divided into three categories as per the superiority and number of microprocessors used. These are:
- Micro computers - Mini computers - Mainframe computers

**35. Distinguish between the three types of computers.**
Ans. A microcomputer is a small computer containing only a single central processing unit (CPU). Their word length varies between 8 and 32 bits and used in small industrial and process control systems. The storage capacity and speed requirements of microcomputers are moderate.
Mini computers are having more storage capacity and more speed than micro computers. Mini computers are used in research, data processing, scientific calculations etc.
Mainframe computers are designed to work at very high speed and they have very high storage capacity. Their word length is typically 64-bits. These are used for research, data processing, graphic applications, etc.



Scanned by CamScanner

**36. In how many ways computer softwares are categorised?**
Ans. Computer softwares are divided into two broad categories—system software and user software.

**37. Explain the two types of softwares.**
Ans. System software is a collection of programs used for creation, preparation and execution of other programs. Editors, assemblers, loaders, linkers, compilers, interpreters, debuggers and OS (operating system) are included in system software.
User software are softwares developed by various users.

**39. What is an editor?**
Ans. An editor is a program and is used for creation and modification of source programs or text. The editor program includes commands which can delete, insert or change lines/ characters.
An editor stores in ASCII form in successive RAM locations of the typed letters/ numbers. This then can be saved by the SAVE command on a floppy or hard disk. The editor can load the program on RAM later for corrections/alterations, if necessary.

**40. What is an OS (operating system) and what are its functions?**
Ans. An operating system provides an interface between the end user and the machine. It performs the function of resource management. By 'resource' is meant a microprocessor, memory or an I/O device.
An OS is a collection of a set of programs that manages machine functions under varied conditions. The different functions performed by the OS include an efficient or effective way of sharing memory, I/Os and the CPU amongst different users. Today, operating systems are DOS, UNIX, WINDOWS, etc.

**41. What are the different types of assemblers used?**
Ans. The different assemblers used are: (a) Macro Assembler, (b) Cross Assembler, (c) Meta Assembler.

**42. What is a linker?**
Ans. A linker is a program that links several small object files to produce one large object file.
A large program is usually divided into several small programs. They are written separately, tested and debugged. The large program is then produced by linking these debugged programmes.
A linker has a link file and a link map. The former contains the binary codes of all the modules which have been combined while the link map stores in it the addresses of all the link files. A linker assigns relative addressing instead of absolute addressing which helps in putting a linker program anywhere in the memory.

**43. What is a locator?**
Ans. A locator is a program which is used to assign specific address when object code is to be loaded into the memory.



Scanned by CamScanner

**44. What are the different assembly languages used for 8085 microprocessor?**
Ans. Two different assembly languages are used for 8085 microprocessor—one is used by the manufacturer INTEL and the other defined in IUL 77.

**45. What is a coprocessor?**
Ans. A coprocessor or an arithmetic coprocessor, as it is so called, performs operations like exponentiation, trigonometric functions, etc. To implement these functions in CPU hardware is a costly one and the software implementation of the above slows down the processor. A coprocessor overcomes these difficulties. The coprocessor has its own instruction set. The CPU and the coprocessor execute their respective instructions from the same program. The instructions belonging to the coprocessor are fetched and decoded by the CPU but executed by the coprocessor.

**49. What is a debugger?**
Ans. A debugger is a program that debugs an object code program by placing it into the system memory and subsequently executing the same.

**50. How does a debugger help in debugging a program?**
Ans. A debugger is a software tool which isolates the problems/drawbacks in a programmer's program.
A debugger helps in debugging a program in the following ways:
(a) A debugger helps in checking the contents of memory locations and various registers and also alter the same if required and rerun the program to check the correctness of the modified program.
(b) Program execution can be stopped after each instruction–hence step by step checking is possible with a debugger.
(c) A debugger can set a breakpoint at any place in a program.
A program then can run up to the breakpoint address and not beyond. Thus any fault in the program up to the breakpoint address can be checked by having a look at the various registers and memory contents. If no fault occurs the breakpoint is set at a latter address in the program and the debugger can again be rerun to check for the correctness of the program. This way the whole program can be corrected by judiciously inserting breakpoints in a program.

**51. What is meant by the term 'word'?**
Ans. A collection of bits is called a word. A word does not have a fixed number of bits—unlike the case of byte (= 8 bits) or a nibble (= 4 bits).
For different microprocessors, the word length varies. For example, for 8-bit microprocessors, the word length is 8-bits while that for 32-bit microprocessors, the word length is 32-bits.
A word is expressed usually in multiples of 2, but no value is excluded. That is, we can have a 19-bit word or 37-bit word, etc.

**52. What is meant by the term 'long word'?**
Ans. The term 'long word' means a group of bits twice the normal length of the microprocessor. Hence, for a 16-bits microprocessor like 8086, a long word means a group consisting of 32-bits.



Scanned by CamScanner